University of Wollongong

Research Online

University of Wollongong Thesis Collection 1954-2016

University of Wollongong Thesis Collections

1996

Neural networks modelling of in-bar width in hot strip mill

Shizhuang Luo
University of Wollongong

Follow this and additional works at: https://ro.uow.edu.au/theses

University of Wollongong

Copyright Warning

You may print or download ONE copy of this document for the purpose of your own research or study. The University does not authorise you to copy, communicate or otherwise make available electronically to any other person any copyright material contained on this site.

You are reminded of the following: This work is copyright. Apart from any use permitted under the Copyright Act 1968, no part of this work may be reproduced by any process, nor may any other exclusive right be exercised, without the permission of the author. Copyright owners are entitled to take legal action against persons who infringe their copyright. A reproduction of material that is protected by copyright may be a copyright infringement. A court may impose penalties and award damages in relation to offences and infringements relating to copyright material. Higher penalties may apply, and higher damages may be awarded, for offences and infringements involving the conversion of material into digital or electronic form.

Unless otherwise indicated, the views expressed in this thesis are those of the author and do not necessarily represent the views of the University of Wollongong.

Recommended Citation

Luo, Shizhuang, Neural networks modelling of in-bar width in hot strip mill, Master of Engineering (Hons.) thesis, Department of Mechanical Engineering, University of Wollongong, 1996. https://ro.uow.edu.au/theses/2521

Research Online is the open access institutional repository for the University of Wollongong. For further information contact the UOW Library: research-pubs@uow.edu.au

NOTE

This online version of the thesis may have different page formatting and pagination from the paper copy held in the University of Wollongong Library.

UNIVERSITY OF WOLLONGONG

COPYRIGHT WARNING

You may print or download ONE copy of this document for the purpose of your own research or study. The University does not authorise you to copy, communicate or otherwise make available electronically to any other person any copyright material contained on this site. You are reminded of the following:

Copyright owners are entitled to take legal action against persons who infringe their copyright. A reproduction of material that is protected by copyright may be a copyright infringement. A court may impose penalties and award damages in relation to offences and infringements relating to copyright material. Higher penalties may apply, and higher damages may be awarded, for offences and infringements involving the conversion of material into digital or electronic form.

Neural Networks Modelling of In-Bar Width in Hot Strip Mill

A thesis submitted in fulfilment of the requirements for the award of the degree of

MASTER OF ENGINEERING (Honours)

from

THE UNIVERSITY OF WOLLONGONG

by

SHIZHUANG LUO, BE

UNIVERSITY OF WOLLONGONG LIBRARY

Department of Mechanical Engineering

1996

DECLARATION

This is to certify that the work presented in this thesis was carried out by the author in the Department of Mechanical Engineering at the University of Wollongong and has not been submitted for a degree to any other university or institution.

Shizhuang Luo

ACKNOWLEDGMENTS

I would like to thank my supervisor, Dr. A. K. Tieu, Associate Professor in the Department of Mechanical Engineering at the University of Wollongong for his supervision and generous assistance and encouragement during the period of this study.

I am grateful to the BHP Steel Flat Products Division (Port Kembla) staffs, Ms. Denise Frances and Mr. Robert Macintosh, Mr. Dan Mistre, Mr. U Nivala, Mr Vladimir and their colleagues for providing me with all my information needs for this thesis.

I would also like to acknowledge the invaluable assistance given by other staff of the department, especially Mr. Tony Kent. I thank other staff who have helped me with my experiments, Mr. Mark MacKenzie and Mr. X. Yao.

ABSTRACT

The steel industry is restructuring to seek higher quality products, rather than a simple increase of the amount of steel produced. Artificial Intelligence provides the necessary tools to improve the current steel processing technology. This thesis deals with the rolling process in a Hot Strip Mill. Width control is considered as a major quality variable, hence this thesis concentrates on Roughing Mill Width Control. Currently, a statistical model is used for width control, which is providing a limited quality performance. Therefore, an accurate model is required in place of any current statistical model.

This thesis presents the research work towards an applicational in-bar width model at a roughing mill. This work is under the Australia Research Council (ARC) collaborative project with BHP steel, Flat Products Division, Hot Strip Mill.

Neural Networks have been proven to be a very good tool for modelling and prediction, for its nonlinear feature, good interpolation ability and adaptability to novelty situation. Hence, a Neural Network was is selected as the tool to perform modelling of the width deformation process. Statistics analysis was carried out to assist improving model accuracy and reduce network redundancy.

Table of Contents

List of Figures

List of Tables

List of Panels

Chapter 1

Introduction to the Rolling Process

With the revolution in Information Science, steel industries have increasingly become aware to the needs of technology upgrade ranging from the making of iron and steel to the final product of diverse purpose, in which every process is modernised to adopt new automatic and high precision equipment. In this thesis, an application of Artificial Intelligence into steel rolling is discussed. To assist understanding it, a comprehensive knowledge of steel rolling processes is required. At this point, the processes of steel production are presented and focus is then given to our specific process problem.

1.1 Hot Strip Rolling Theory

Although there are many ways to obtain final steel products, the most common and accurate way is through rolling, although recently strip casting is gaining popularity. Rolling can be done in either normal temperature, or hot temperature. The first one is referred to as cold rolling, which normally produces very thin strips. The latter one is refer to as hot rolling, which involves large thickness reduction. Hot rolling is done at high temperature 1000- 1200 °C, when the steel is very "soft". There is less deformation resistance compared to that at normal temperature.

1.1.1 Basic rolling theory

To assist understanding of rolling theory this report will first discuss general rolling theory, then come to a more specific hot-rolling theory.

The first major step in the flat processing of steel involves the conversion of slabs into hot-rolled strips. At high temperatures, the slabs are in the austenitic state and relatively easy to deform.

In normal rolling process, slabs go through the work rolls and are deformed by the rolling pressure. For a rolling process in an ideal condition, the followings are hypothesis about the rolling:

(i) two work rolls are being driven;

(ii) the work rolls have the same diameter;

(iii) the work rolls have the same rotational speed;

(iv) the workpiece maintains a constant velocity;

(v) no other forces on rolled slab except for the rolling pressure from work rolls;

(vi) the workpiece material is isotropic and homogenous;

(vii) the effects of the strain hardening and strain rate are neglected.

Geometric deformation caused by rolling

When rolled, the workpiece has plastic deformation in three dimensions, as shown in Fig. 1.1, that is the workpiece thickness is reduced from h_0 to h_1 by a ratio $\eta = h_1 / h_0$. η is called draft coefficient. The workpiece width changes from b_0 to b_1 according to a spread coefficient $\beta = b_1 / b_0$. The workpiece length increases from l_0 to l_1, with an elongation coefficient $\lambda = l_1 / l_0$. The geometric relation is

$$h_0 b_0 l_0 = h_1 b_1 l_1$$

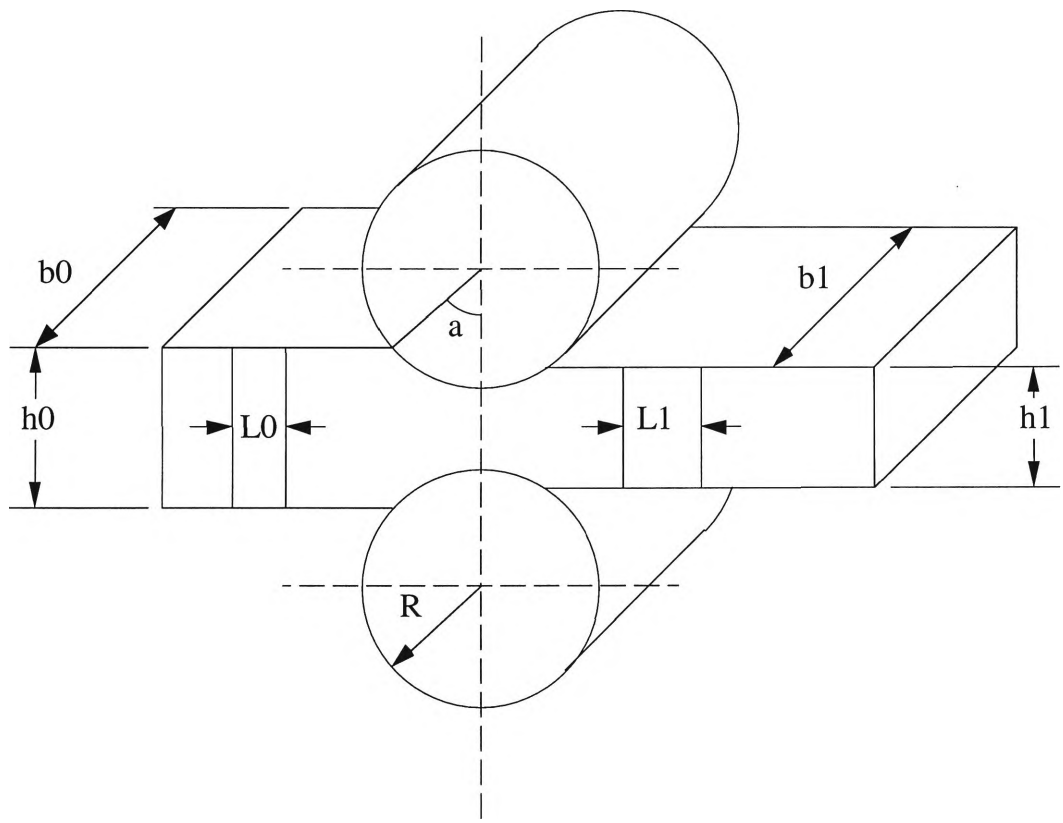


Fig. 1.1 The geometric relationships in a rolling process

Only when the workpiece enters the deformation zone can the rolling process be established. The workpiece bite condition can be phased into roll bite and biting phase in the arc of contact.

At roll bite, when the workpiece touches the rolling work roll, there is a positive pressure N and friction force as illustrated in Fig. 1.2. The relationship is described as

$$N \sin\alpha - T\cos\alpha = 0$$

The critical condition then is the friction coefficient $\mu = \tan\alpha$

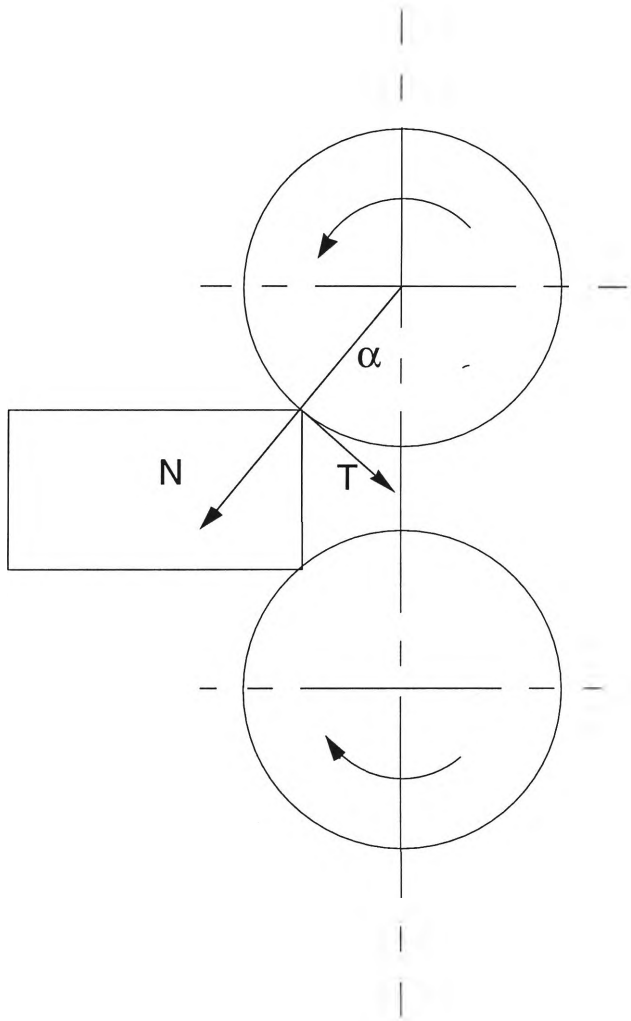


Fig 1.2 Rolling Biting Process

For rolling to occur, it is required that $\mu > \tan\alpha$.

1.1.2 Hot Strip Rolling

The most important feature of hot rolling is that it can yield great dimensional deformation. At the same time, it goes through some microstructure changes.

In the manufacture of flat-rolled products, certain physical and metallurgical properties such as yield strength, tensile strength, ductility and formability (stretchability and average strain ratio) are sought for the various products (Pitsch and Hougardy, Dahl, and Schauwinhold and Schluter, 1984). These may be achieved, in part, by the selection of various chemistry in steel making. However, the properties of a steel are also dependent on its microstructure. This, in turn, is dependent on the thermo mechanical processing of the steel and the various solid-state reactions that occurs during processing. (Ouchi et al., 1982)

In heating the slab, the body-centered-cubic (BCC) structure existing at ambient temperatures is changed to an austenitic, face-centered-cubic (FCC) structure when the AR3 transformation temperature is exceeded. The grain size, ranges from 10 to 1000 microns. Care must be taken to avoid excessive surface temperature otherwise grain coarsening may occur.

Roughing Mill rolling refines the microstructure by its breakdown during deformation and by recrystalisation between passes. The final size of the recrystalised grains

resulting from multiple recrystalisations depends upon several deformation parameters. It decreases as the deformation temperature decreases, reduction per pass increases and strain rate increases.

In the Finishing Mill, below the recrystalisation temperature range, the austenite grains flatten in the thickness direction and elongate in the rolling. To obtain enhanced strength, toughness and formability, austenite grain flattening in the final pass is desirable.

The run-out table cooling and coiling will dramatically influence the microstructure change by carefully selecting the cooling temperature and cooling rate.

1.2 Rolling equipment and control

1.2.1 Hot Strip Rolling Production Arrangement

Hot rolling is a vital step towards final product of steel. In the processes, slabs from the continuous casting process are reheated and then rolled to the dimension as required by the customers. The slabs in general are larger in thickness and shorter in length than the final strip. To achieve such changes in dimensions, it requires many rolling phases. To illustrate the procedures, major steps of hot strip mill rolling are shown in Fig 1.3.

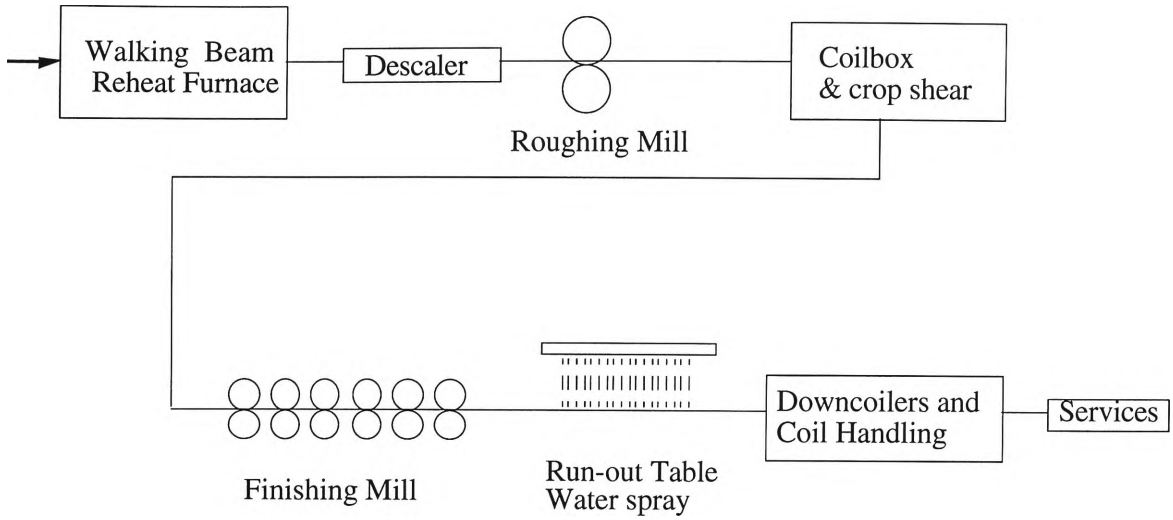


Fig 1.3 Hot Strip Mill Production Procedures

The purpose of the walking beam furnace is to reheat the slabs to appropriate rolling temperatures (usually extracted at 1225 °C). The walking beam method carries the slabs through the furnace, replacing the pusher type method, thereby eliminating slab surface and cold spots gouging. An extractor slab discharge replaces the gravity discharge used in the old furnaces, thereby eliminating edge damage of the slabs.

The Roughing Mill area includes from the descaling station to the entry of the Coilbox. The slabs from the reheat furnace, after going through the descaling, come to the roughing mills by the transport of table rolls. In Roughing Mills, slabs thickness is decreased by rolling reduction. . A typical Roughing Mill operation reduces a 230 mm thick slab to approximately a 30 mm thick-transfer bar by means of a conventional 7-9 pass drafting practice. The roughing mill also controls the major portion of the final width from hot strip rolling. There are two major categories production arrangements around the world. One is tandem roughing mills with usually more than 4 roughing mills. The advantage of this kind of mill, is that it can have high production efficiency as production can be a continuous streamline. The other one is reversing roughing mills. This kind of arrangement saves production space and heat loss from slabs. There are

also roughing mills mixed with the above two kinds of arrangements, that is, it has a tandem roughing mill and 2-stand reversing mills.

The Coilbox concept was developed by Stelco Inc of Canada during the 1960's. The first unit was installed at Stelco's Hilton works in 1972. Upon leaving the mill after the last pass, the bar is directed into the coilbox where it is formed into a tight coil. The coiling operation is then reversed and the new head end of the bar is fed into the Finishing Mill. That is, the tail end becomes the new head end and the bottom of the bar extracted from the rougher becomes the top of the bar through the finishing train.

This technique has many advantages as it can improve production quality by delivering the bar with more uniformed temperature, save energy and increase slab mass. Before the transfer bar enters the Finishing Mill, the head and tail are cut by the crop shear.

The Finishing Mill takes the intermediate thickness transfer bar from the Roughing Mill and Coilbox, and rolls it to the final desired thickness using 5 to 7 four-high stands. With hot rolled strips coming out of the Finishing Mill, they are coiled in the Downcoilers and then transferred to storage area.

1.2.2 Roughing Mill and its control system

The Roughing Mill is of particular concern to us as it is the research area of this thesis. As mentioned in previous section, there are many types of Roughing Mills. This thesis deals with a two-stand four-high mill, which has a roughing edger and horizontal

reversing mill. Slabs go through this roughing mill usually in 7 passes, but 9 passes are required for hard steel grades.

The roughing edger (RE) is a 2-high mill with two rolls vertical. It has the function of controlling the bar width by shifting the gap between the work rolls. The reversing mill (or called roughing mill rougher, RR) is a 4-high horizontal mill. The draft in all passes reduces the bar thickness dramatically. The reversing mill controls the thickness of transfer bar.

The 4-high mill is made up from two work rolls with smaller diameters and two backup rolls with larger diameter. The backup rolls can support the workrolls and reduce the bending. Usually a four high rolling mill is driven through work rolls, while backup rolls are driven by friction forces.

Although the roughing mill is said to control dimensions roughly, it is actually crucial for the final product quality, especially for strip width control. There is still an accurate thickness control system in the following finishing mill, but a smaller width control after roughing mill. If anything goes wrong in width in the roughing mill, it will affect the final strip width.

The current control system structure for the roughing mill in BHP SPPD HSM is shown in Fig. 1.4. The central control system is the highest rank in the control

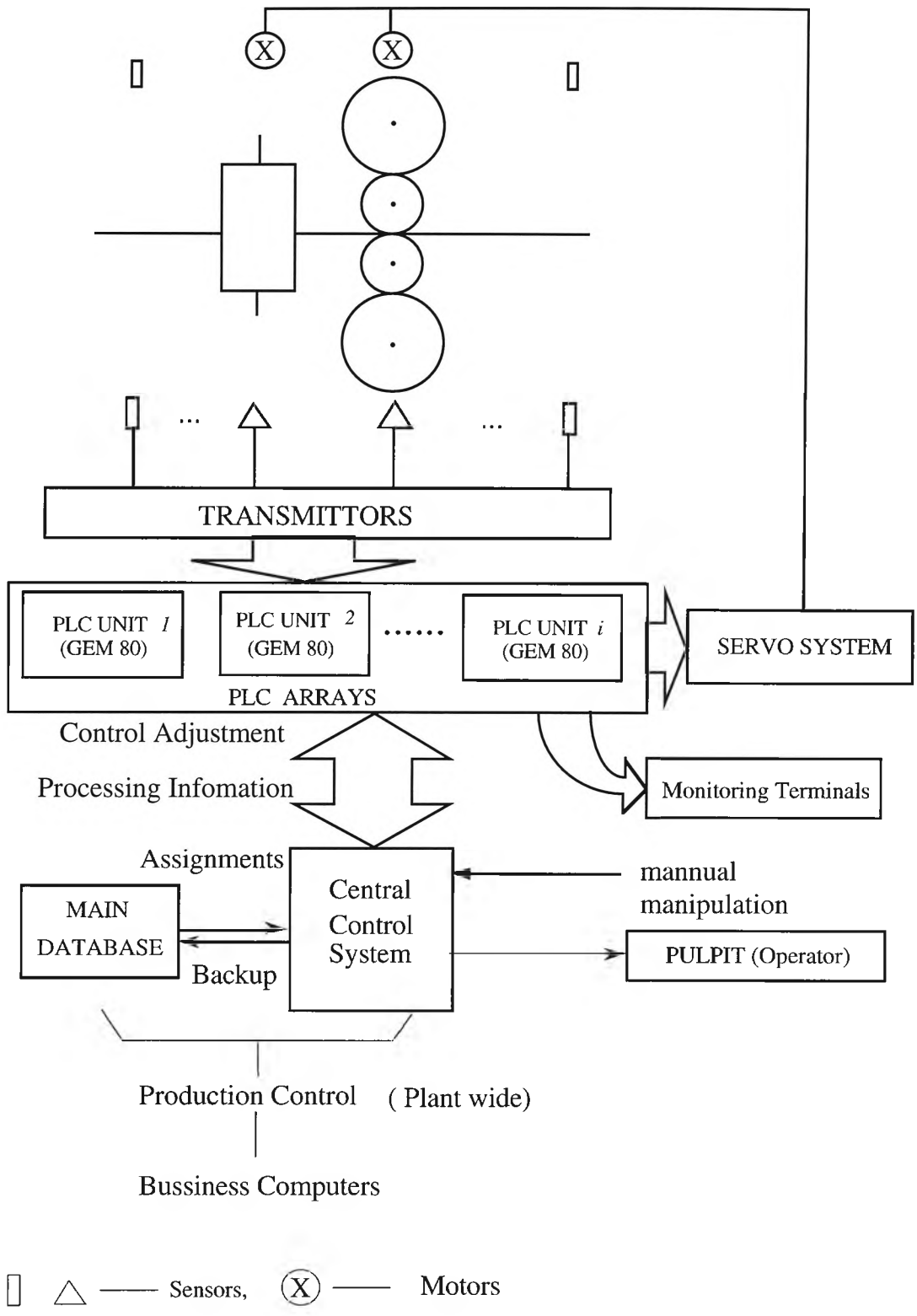


Fig 1.4 Control System for Roughing Mill

hierarchy. It receives data from data base and other computers about production schedules and presets control aims for roughing mill. It allows the operator to impose certain changes on the mill. The PLC arrays are the controllers that are programmed to respond to higher system commands and signals from sensors. PLCs then operate the servo system to perform draft on rolling mills and other control variables, e.g. roll gap, motor speeds, thus controlling the dimensions of the bars being rolled.

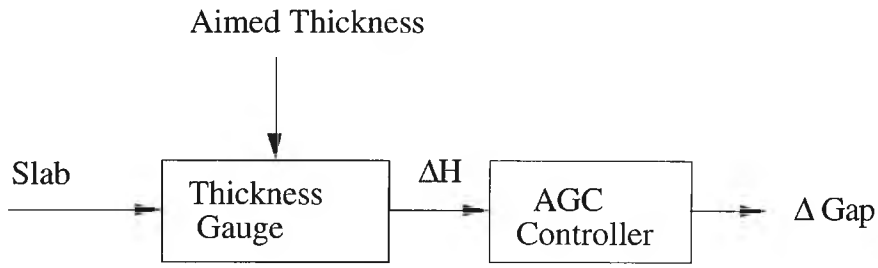


Fig. 1.5 AGC Feed-forward System

On the roughing mill, there are several specific controls. The primary one is the Automatic Gauge Control (AGC), which control the exit thickness of bars from roughing mill. There are two kinds of AGC systems, Feed Forward AGC and Feed-Back AGC. The Feed-Forward AGC measures the thickness of slab before it enters into the mill and the Feed-forward AGC controls the roll gap. Illustrated in Fig 1.5, it is a simple Feed-forward control. Just the opposite, feed-back AGC operates as a typical feedback control system. The PLCs take the gauge value at the exit end and calculate the error between the measured and the aimed thickness. It can take into account the delay factor and give a control signal to adjust the gap in order to achieve zero error. This is shown in Fig. 1.6.

Similar to the AGC is Automatic Width Control (AWC). The AWC systems are designed to accurately control the width. The AWC in BHP SPPD controls the head and

tail width as well as the mean bar width. There is also a Feed-forward control system for in-bar width control. However, width is not accurately controlled like thickness. Also, there are many drawbacks in the AWC system, which will be discussed this in section 1.3.

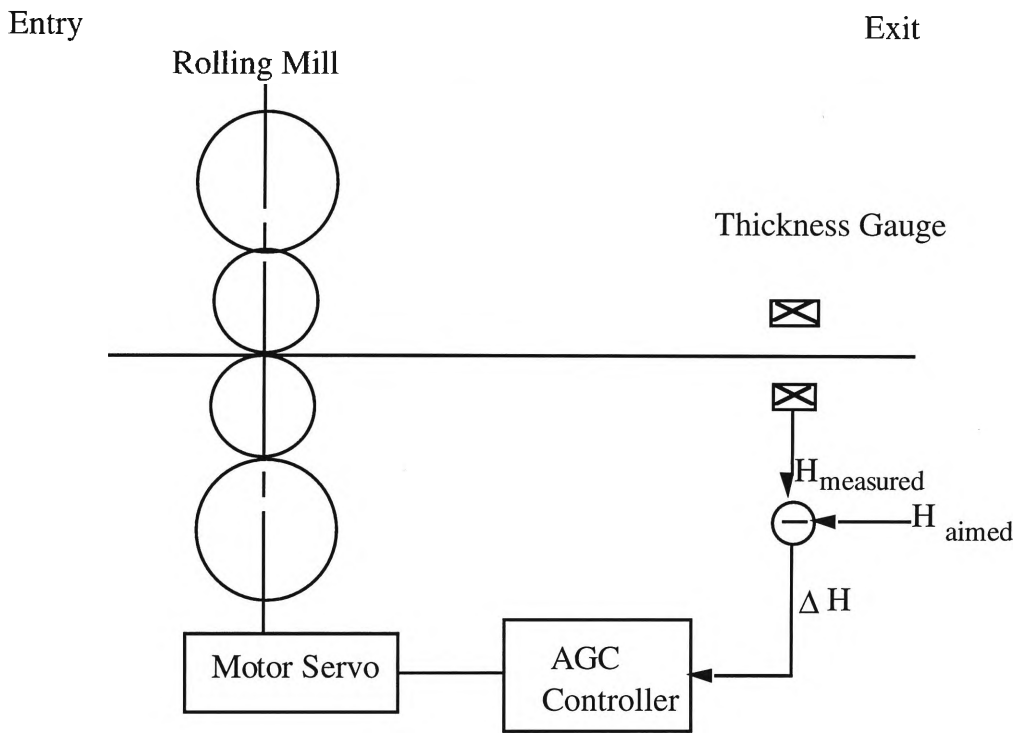


Fig. 1.6 AGC Feed-back Control System

1.2.3 Finishing Mills and its control systems

The Finishing Mill is the section where all the strips are rolled to final dimensions and then taken into coils. Modern Finishing Mills usually adopt tandem continuous arrangement, with 5-7 or more mills along the production line. This way of continuous rolling has the advantage of high rolling speed, automatic thickness and profile control

with high accuracy. All mills are 4-high mills with hydraulic screw down-system. The continuous rolling has its own theory in control.

To set up and maintain the continuous rolling, the metal flow at every mill has to be satisfied. That is

$F_1V_1 = F_2V_2 = \ldots = F_nV_n, \qquad mm^3/s$

1,2,...,n	mill sequence number
$F_1, F_2, ..., F_n$	workpiece transverse section at each mill
$V_1, V_2, ..., V_n$	rolling speed when workpiece pass each mill
$F_1V_1, F_2V_2, ..., F_nV_n$	workpiece metal flow at each mill

Once this condition is broken, the continuous rolling will be out of balance.

There are many control hardware installed in the Finishing Mill. The control systems of finishing mill usually include X-ray gauges at the exit of finishing mill, hydraulic capsules, work roll side shifting, work roll bending equipment and speed control system. X-ray gauges provide thickness measurement which is used in a feed back control. Hydraulic systems are the servo systems for adjustment. Work roll bending and side shifting are for the strip profile and flatness control. Speed control is to ensure that condition of continuous rolling are maintained and it provides control of width through interstand tension. There could also be feed forward thickness control from one mill to the one after it. Fig 1.7 is a typical Finishing Mill control system.

3 0009 03201054 3

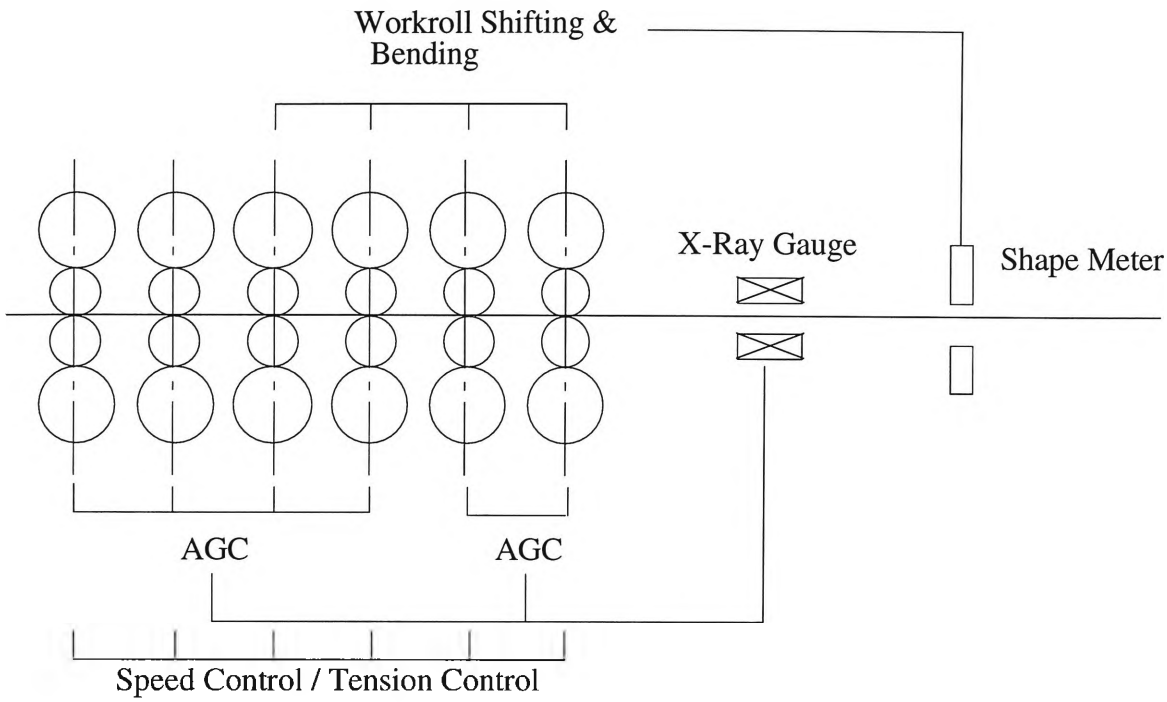


Fig. 1.7 A Typical Finish Mill Control System

1.3 Problems and expectation

The steel industry is facing the need for increased automation to improve product quality and productivity. The processing of steel is becoming more competitive as all the steel works are upgrading their plants. Hence the quality of steel product, is more important than ever in the steel industry, as this will affect its market share and reputation. One of the ways to achieve this goal is to install new hardware, such as new rolling mills or new control equipment. In 1991 alone, there have been major changes in steel plants in US and Canada when many updated equipments were installed (Labee and Samways, 1992). Another way to improve product quality is to implement more control systems onto current rolling mills, with the latest development in artificial intelligence. It is relatively very low cost with significant improvement on product quality.

This thesis deals with the roughing mill in particular. At the Roughing Mill, the most common rolling is the 7-pass rolling, with pass 1, pass 3, pass 5 and pass 7 being forward passes, and the others being reversing passes. The slabs change dimensions gradually pass by pass. At the exit of pass 7, transfer bars are produced with certain dimension, which hopefully can meet the requirement of aimed thickness and width.

There are sensors mounted on the Rouging Mill transmitting useful to the mill control system. As mentioned in 1.2.2, there are both PLC control and Central control system installed on the Roughing Mill. Specific control subjects in Roughing Mill are AGC and AWC. The AGC has the same problem of stability and accuracy as any other feed-forward or feed-back control system. For the AWC system, there are several questions.

Width control is divided into three areas. The first is head and tail stroking control, the second is bar to bar width control, and thirdly in-bar width control. Head and tail stroking is about the width control of one metre from the bar heading edge and from the bar tailing edge. Bar to bar width control is about the mean width control of slabs. In-bar width control is the width control of the bar's body. The in-bar width determines the quality of a finishing coil, and this is the focus of the research for this thesis.

Current system is a feed-forward system. At pass 1, it adjusts values from a Width gauge (WG0) at exit from the reheat furnace after descaling. It has a very simple formula to adjust the gap of Roughing Edger according to the width deviation at WG0. The purpose of this adjustment is to compensate for its deviation from the mean value. Similar adjustment occurs in pass 3, 5 and 7, which takes the width value from the exit width of pass 2, 4, 6.

There is no physical model for such feed forward control. It calculates solely on the basis of input width. In real rolling process, however, many other factors contributes to width deformation, such as rolling temperature, slab grade and etc. It is then not unusual that such a feed forward control system performs poorly sometimes and has to rely on an adaptive scheme to improve the accuracy of the body width. The in-bar width variance is out of range sometimes. This indicates a need to incorporate a new accurate model to predict the in-bar body width.

The research topic in this thesis will deal with in-bar model, how to use neural network to build up and analyse a model.

It has been specified by BHP Steel that improvement of the roughing mill control and rolling mill are not to be considered here. All the possible solution lead to the question, how to improve the control system by either implementing a new algorithm or adding some new control strategy. Artificial intelligence (AI) is then our first choice. AI has evolved so much since it is first introduced in 1950's. Especially, it has begun to boom since the late 80's. Now, artificial intelligence offers large number of applications from management, control algorithm to system modeling. Neural networks, fuzzy logic and expert system, which are part of Artificial Intelligence, can all be used in improving the in-bar width control.

Chapter 2

Literature Review

2.1 History and development of Artificial Intelligence

Artificial Intelligence (AI) attempts to understand intelligent entities. Here we adopt a definition from Charniak and McDermott (1985) that AI is

The study of mental faculties through the use of computational models.

AI addresses one of the ultimate puzzles. How is it possible for a brain to perceive, understand, predict and manipulate a world far larger and more complicated than itself?

AI is one of the newest disciplines, which was formally initiated in 1956. Although a young field, it has inherited ideas, viewpoints and techniques from other disciplines. For over 2000 years, philosophers have tried to understand how seeing, learning, remembering could, or should be done.

For artificial intelligence to succeed, two things are essential: intelligence and artifact. The computer has been unanimously acclaimed as the artifact with the best chance of demonstrating intelligence.

The first AI work was done by Warren McCulloch and Walter Pitts (1943). They showed that any computable function could be carried out by some network of connected neurons, and all the logical connectives could be implemented by simple net structure. Dornald Hebb (1949) demonstrated how learning takes place. In the early 1950s, Claude Shannon (1950) and Alan Turing (1953) were writing chess programs for von Neumann-style conventional computers. Other researchers at that time, such as Marvin Minsky, Dean Edmonds and John McCarthy, all contributed to the advance of Artificial Intelligence. The term of Artificial Intelligence was coined by John McCarthy during the two-month workshop held in Dartmouth College in 1956.

In the early stages (1952-1969), Allen Newell and Herber Simon developed a reasoning program, the Logic Theorist (LT). It was followed up with the General Problem Solver, or GPS, which was designed from the start to imitate human problem-solving protocols. John McCarthy did a lot as he developed LISP, which is the main language used in AI. The program called Advice Taker can be seen as the first complete AI program. The work of Winograd and Cowan (1963) showed how a large number of elements could collectively represent an individual concept, with a corresponding increase in robustness

and parallelism. Bernie Widrow (Widrow and Hoff, 1960; Widrow, 1962), developed networks Adalines. Frank Rosenblatt (1962) formulated his perceptrons and proved the convergence of perceptrons.

In this period, it was very optimistic that AI can be the solution to all problems. Before, people had not considered that a machine or computer can think. It was taken as an arithmetic tool. However, after the failure of machine translation, AI came to a low point. Practically, the existing computer performance at that time could not satisfy the need of AI. This period was marked by some publications (Lighthill, 1973 and Minsky and Papert, 1969). These publications were critical and pointed out the limit of perceptron.

In the early 70s, one area of AI became the focus of study, which is the so called **expert system**, or **Knowledge-based system**. The Dendral program (Buchanan et al., 1969), was an early example. The widespread growth of applications to real-world problems led the development of several representation languages, such as Prolog. However, Minsky's frames (1975) is a more structured approach, collecting facts and arranging the types into a large taxonomic hierarchy analogous to a biological taxonomy. This field came to a boom in 1980 and is still growing steadily.

The field of neural network was neglected until the study by Hopfield (1982). He used techniques from statistical mechanics to analyse the storage and optimisation properties

of networks, leading to a significant cross-fertilization of ideas. After the rediscovery of multi-layered perceptron, or back-propagation network, the use of neural network became promising than ever. Its beginning was marked by the publication of *Distributed Processing* (Rumelhart and McClellend, 1986). Now neural networks are being used extensively in all area of studies

Academically speaking, Fuzzy logic is not a field of AI. However, it has its root from probabilistic reasoning, which is the heuristic part of AI. Hence, fuzzy logic is sometimes taken as part of artificial intelligence. It tried to mimics the way of human logistic expression with uncertainty. Since the first introduction of fuzzy concept, fuzzy logic has been applied to many industrial applications. The most outstanding use is the fuzzy logic controller (FLC), which greatly enhanced the conventional controller performance.

2.2 AI in the steel industry

Artificial intelligence has been applied to steel industry for more than 10 years in all the areas from iron making to steel processing. Below is the brief introduction of some AI applications in the steel industry.

2.2.1 Expert system in steel industry

As early as 1985, Fischer (1985) from Inland Steel Co. gave his ideas of how to build an AI system, or an expert system. He described the concept of knowledge systems and its components, knowledge, skill and reasoning. The application was a knowledge-based system for scheduling galvanizing lines. It evolved from problem description, design of assistant to the final advanced knowledge-based assistant (expert). The structure and conceptual design of such knowledge system were more important as it allows optimal features and expansion.

Agapi Svolou and John Hudak (1987), supported by the Association of Iron and Steel Engineers, developed a prototype "expert diagnostic assistant" for a multi-stand, hot strip mill. This particular prototype addresses the operational aspects of the six finishing stands and the effects the stands have on product quality such as thickness and width. The purpose of the prototype was to demonstrate how an expert system could be used to assist an operator or relatively untrained technician, in fault analysis of a rolling mill. They expected that the use of such a expert system MOLE, would suit typical problems encountered in steel mills, namely, loss of skilled people due to retirement and a long job apprenticeship lasting over ten years.

The slab assignment expert system in Sumitomo Metal Industries (Jimichi, et al. 1990), was developed with an expert system building tool SMI/Marks-II. With the system, the

number of assigned pairs is almost equal to that by the expert. Assignments of slabs with excessive grades could be eliminated by the system. A similar system by Konishi, M., et al. (1992) was also developed.

In the Wakayama Steel Works of Sumitomo Metal Industries, Ltd., an on-line type expert system (Nakamura et al., 1992) was introduced in 1988 for the purpose of adding an overall diagnosis function to the On-line Facilities Monitoring system. This expert system is utilized as a tool which supplements deficiencies in experience and expert knowledge, supports reductions in maintenance time, and has a technology transfer function.

AI (expert systems) had been applied in many production fields in Sumitomo Metal Industries (Yokoi et al., 1992). Expert system/knowledge system had been used from design, planning to process control and diagnosis. Systems installed were:

- Support for casting and melting sheets in steel making plant
- Support for large-sized pipe material design
- Material arrangement planning in the hot strip mill
- Assignment for excessive steel bloom order
- Management of blast furnace conditions
- Mould melt level control in continuous casting
- Coil transfer scheduling in hot strip mill
- Remote control facility,

- Problem diagnosis for hydraulic screw-down in cold strip tandem mill
- Support for automatic operation by computer.

With more expert systems being used in industry, commercial expert systems have been developed for applications in the steel industry. Those systems allow other features to be integrated with the knowledge system, such as fuzzy logic. Gaither (1991) described the integration of rule-based expert systems and fuzzy logic in process control application. Schnelle and Mah (1992) used G2 expert system shell to generate their real time expert system for quality control. Their system is a plant-wide on-line monitoring system. It monitored steel products as they move through a mill and helped to produce products that met customer's quality standard. It made use of information about the plant and used the inference engine to obtain a possible conclusion.

An expert system is a software that mimics the reasoning capability of a human through the use of an expert knowledge base. Such systems take advantage of the conventional techniques in creation of decision support and operator advice systems. An important development of this system is a real-time expert system (RTES) that enables the expert knowledge base to be operated on-line. A significant type of RTES is the Cogsy system, which is being used by the steel industry and in particular Avesta Sheffield (Burras, 1995) to increase production.

With many expert systems in production, Cohen (1994) gave his advice on implementing expert system on rolling mills. It would insure the system success.

2.2.2 Fuzzy logic used in steel industry

Hasegawa and Taki (1991) from Nagoya Works of Nippon Steel developed a shape control system for its No. 3 cold strip mill. The actual shape of strip being rolled is detected by a shapemeter and is feedback controlled by work roll bending, screwdown leveling, and coolant zone control. The shape control system is characterized by the application of fuzzy set theory in improving control accuracy. Its control performance is good for complex shapes as well as simple shapes.

A method for determining the optimum forming paths in the roll bending process of plates by combining the finite element simulation with the fuzzy reasoning is proposed. In this method, the forming conditions in the finite element simulation are automatically changed to attain the desired shape of the plate with the help of the fuzzy reasoning by monitoring the calculated results such as the curvature of the plate, the stress and strain distributions, etc. in each deformation step. To form the front and the rear ends of a plate into a proper curvature by a three-roll bending process, it is essential to give an appropriate path of the top roll. For this purpose the optimum motion of the top roll is successively searched for by the finite element simulation. Fuzzy reasoning is used to

determine the motion of the top roll by taking account of the effects of the plate thickness, the flow stress and the difference between the desired and the calculated curvatures. It is shown that the decrease in the curvature near both ends of the bent plate is reduced by the use of the method by Osakada et al. (1993)

In the study by Jung et al. (1995) fuzzy theory was introduced in controlling the strip shape in cold rolling. Fuzzy control algorithm is developed based on the production data and operational knowledge. The cold-rolled products are characterized into several types based on its irregularity, 'edge wave', 'center buckle', 'W-type', and 'M-type'. The developed fuzzy controller calculates for each type of irregular strip shape, the change of bending forces of work and intermediate rolls, using fuzzy rules and fuzzy inference. To simulate the continuous shape control, the fuzzy controller was combined with an emulator which was developed using neural network. The developed fuzzy controller and emulator simulate the cold rolling process until the irregularities converge to a tolerable range to produce a uniform cross-sectional shape. The results from the various simulations demonstrate that the developed fuzzy control algorithm works for various types of irregular cross-sectional shapes as well as simple shapes.

The improvement of accuracy of head-end strip thickness is required strongly these days. But generally speaking, it is very difficult to maintain accurately under the production of small-lot orders for many different types of products. As one of the positive countermeasures to these difficulties, on-line adaptive methods had been

developed at Yawata's hot strip mill, Nippon Steel Corp(Oda, 1995). The main idea was based on improvement of prediction accuracy of rolling forces, by means of the forward slip measuring system, in order to estimate the coefficient of friction and deformation resistance independently. Further correction method had been introduced employing fuzzy theory which was a dynamic gap control by finding a certain relationship between the predicted rolling force and the measured one. Satisfactory results were obtained and it was believed that this approach was applicable for a gauge control method under increasingly diversified production style.

2.2.3 Neural networks used in Steel Industry

The research by Desrochers and Saridis (1980) adopted the idea of learning theory of neural networks. They presented roll force control methods to be used with the predictive force setup model of the finishing stands in a hot rolling mill. The mill practices achieved a desired strip gauge by using a predictive force model to set-up the roll gaps on the finishing stands. The decision process of the operator was modelled by pattern recognition methods to obtain this extra degree of feed-foward control. Feedback control was provided from one steel run to the next by an adaptive controller which used a linear reinforcement learning scheme to adjust its parameters.

Hitachi, Ltd. had developed entirely pattern recognition and control techniques to recognize spatially distributed waveform patterns, and to operate multiple final control

elements for automatic shape control of cold rolling mills (Hattori et al., 1993). Conventionally, skilled operators recognize and manually control waveform patterns based on sense and experience. The new technique recognizes and controls patterns by means of neural networks and fuzzy logic to realize full automatic shape control of a Sendzimir Rolling Mill. Shape control with this type of rolling mill is difficult with conventional automatic control systems because of complicated rolling phenomena and the difficulty of making a control model. Tests with an actual machine proved that the new technique achieves more accurate control than the conventional manual operation by skilled operators.

Another paper by Hattori et al. (1992) mentioned their neural network system for flatness control of a cold rolling process. The network was used to simulate the skilled operator, while a fuzzy control algorithm were used for operation.

Classification of spatially distributed measurements in industrial processes is often a difficult task. A typical example is given by the interpretation of the gas temperatures from horizontal probes in blast furnaces. Bulsari and Saxen (1995) presented a neural network-based classification of temperature measurements from an above-burden probe. An expert on blast furnace supervision and control first classified a large set of temperature profiles into six different stereotype patterns. Roughly 40% of this material was used for training feedforward neural networks, while the remaining profiles were used for evaluation of the networks' performance in order to determine an

appropriate network size. In general, consensus was found among the networks in that similar erroneous or inconsistent classifications made by the human expert were detected. However, the network size clearly affected the quality of the classifications. The work demonstrated the merits of a rapid, consistent, and automatic (neural) interpretation, as well as the risk of misclassification and the subjectivity involved when human experts have to evaluate complex patterns.

In Avesta Sheffield's steelmaking shop, expert systems were trained, rather than programmed, to extract relevant information from a mass of plant data and present it to the operator in an advisory form. Three applications are described by Burras (1995).

In the case of mill automation, the use of neural networks is advantageous, at very the least, in describing the technical process. The use of neural networks at the process control level for the finishing mill section of a wide hot strip mill was discussed (Portmann et al. 1995). The neural network must take over some of the functions of the process models as well as the model adaptation.

In Sweden, artificial neural network programmes are being applied to temper rolling, hot strip rolling and a cluster mill (Leven et al, 1995). Predicted results are within 2-3% of more rigorous FEM calculations for temper rolling and within 5% for the HSM and cluster mill.

The Intelligent Arc Furnace Controller (Staib and Bliss, 1995) has three neural network models: the first one predicts furnace operations, the second one works as a traditional controller and the third one optimizes the operation of the furnace. With this system up to 8% of the electric power and up to 25% of electrode consumption can be saved. Productivity increases of 12% have been achieved. There described also a neural network used on arc furnace operation at Birmingham steel (Anon, 1996).

Back-propagation neural networks are utilized to store and predict the flow stresses of several steels (Hwu et al., 1996). Similar application was carried out by Rao and Prasad (1995). In other cases, on-line implementation of an ANN at a temper mill plant has led to a hybrid model that mixes symbolic and connectionist modules (Pican et al., 1996). In research by Auzinger et al (1995), material hardness is adapted by the method of on-line recursive regression analysis. Its dependence on chemical analysis, temperature and reduction is derived by means of a neural network. Schmitter(1995) used neural network in grain size determination and classification of iron carbides.

Dom (1996) presented an expert system that acts as a consultant to help making production decisions more transparent. Steelmaking involves a variety of complex processes which do not lend themselves to exact mathematical modeling. Steel manufacturers often rely on the experience of individual experts to provide reasoning based on incomplete and uncertain data. The steel industry has adopted expert systems relatively early for further automated production. Although quality optimisation and

energy conservation are important aims, the most important motivation for applying expert systems seems to be production standardization.

Sun , X. et al. (1995)presented a new way to identify the load distribution for hot-strip finishing mill by artificial neural networks. It has the advantage of on-line calculation with strong learning function and precise results. It has overcome the shortcoming of traditional methods.

In Maheral et al.'s attempt (1995) to get around the real-time impasse associated with a conventional numerical approach to predictive modelling, an integrated AI technique had been proposed and its validity has been demonstrated. Hybrid in nature, their approach combined a 'bottom-up' connectionist paradigm with a top-down real-time knowledge-base system. The immediate goal was to demonstrate the application of these techniques to specific aspects of the actual, albeit small scale, hot steel rolling facilities. The neural networks were trained on a mixture of experimentally gathered data and there generated from mathematical models. Neural networks predicted the temperature behaviour of a hot-steel slab during run-out cooling. Based on industry data, the system discussed in the paper was able to predict the final thickness, roll separation force, and the springback of the steel slab. Furthermore, taking the mill's loading capacity into account, a hybrid real-time knowledge-based/neural network system generated the rolling schedule needed to produce a strip of steel of a specific gauge from a slab of a given composition, initial thickness and temperature.

2.3 Summary

The research works listed above are AI applications in iron and steel industry. These surveys cover a broad range of the areas, such as, operational research, production monitoring, process control, fault diagnosis, which results from the iron-making, the blast furnace, continuous casting, hot strip rolling to cold rolling.

The surveys revealed that Neural Network are increasingly being used in hot and cold strip mills. However, there are still many unknowns with complex processes, which need to improve through the application of AI or fuzzy logic.

The literature survey shows a lot of applications in rolling mills to improve such matter as are on shape, flow stress control, and thickness control. However, there is no literature published on the use of AI in width control, which is a very important quality of hot rolled product. Indicating that there is a need to apply neural networks in this area.

Chapter 3

Neural Networks Modelling Theory

As the most promising branch of Artificial Intelligence, Neural Networks have been applied to many diverse areas, such as modelling, pattern recognition, knowledge base, signal processing, and computer graphic processing. Here, we are going to use Back-propagation neural networks as a modelling tool in Roughing Mill width control.

The use of mathematics to simulate and to make predictions about the real world has a long and distinguished record in the physical science; so much so that mathematics has become the basic language of physics and its application in engineering. Neural networks is just a kind of mathematics which simulate human thinking process. In this chapter, the theoretical interpretation of Neural Networks is introduced first. Its structures and algorithms are discussed. Following it is the convergence and some other aspects concerning the use of neural network.

3.1 What is a Neural Network?

The concept of Neural Network results from its biological origin. It is regarded as a mathematical model to simulate operation of the brain. The simple arithmetic computing elements corresponds to neurons - the cells that perform information processing in the brain - and the network as a whole corresponds to a collection of interconnected neurons. For this reason, the networks are called neural networks.

3.1.1 The concept of Neural Network

Ramo'n y Caja'l (1911) first introduced the idea of neuron as structure constituents of the brain in his pioneering work to understand the brain. Now we know that the neuron, or nerve cell, is the fundamental functional unit of all nervous system tissues, including the brain. Each neuron consists of a cell body, or soma, that contains a cell nucleus. Branching out from the cell body are a number of fibres called dendrites and a single long fiber called axon. Dendrites branch into a bushy network around the cell, whereas the axon stretches out for a long distance-usually about a centimetre (100 times the diameter of the cell body). Eventually, the axon also branches into strands and substrands that connect to the dendrites and cell bodies of other neurons. The connection junction is called a synapse. Each neuron forms synapses with anywhere from a dozen to a hundred thousand other neurons. Figure 3.1 shows the parts of a neuron.

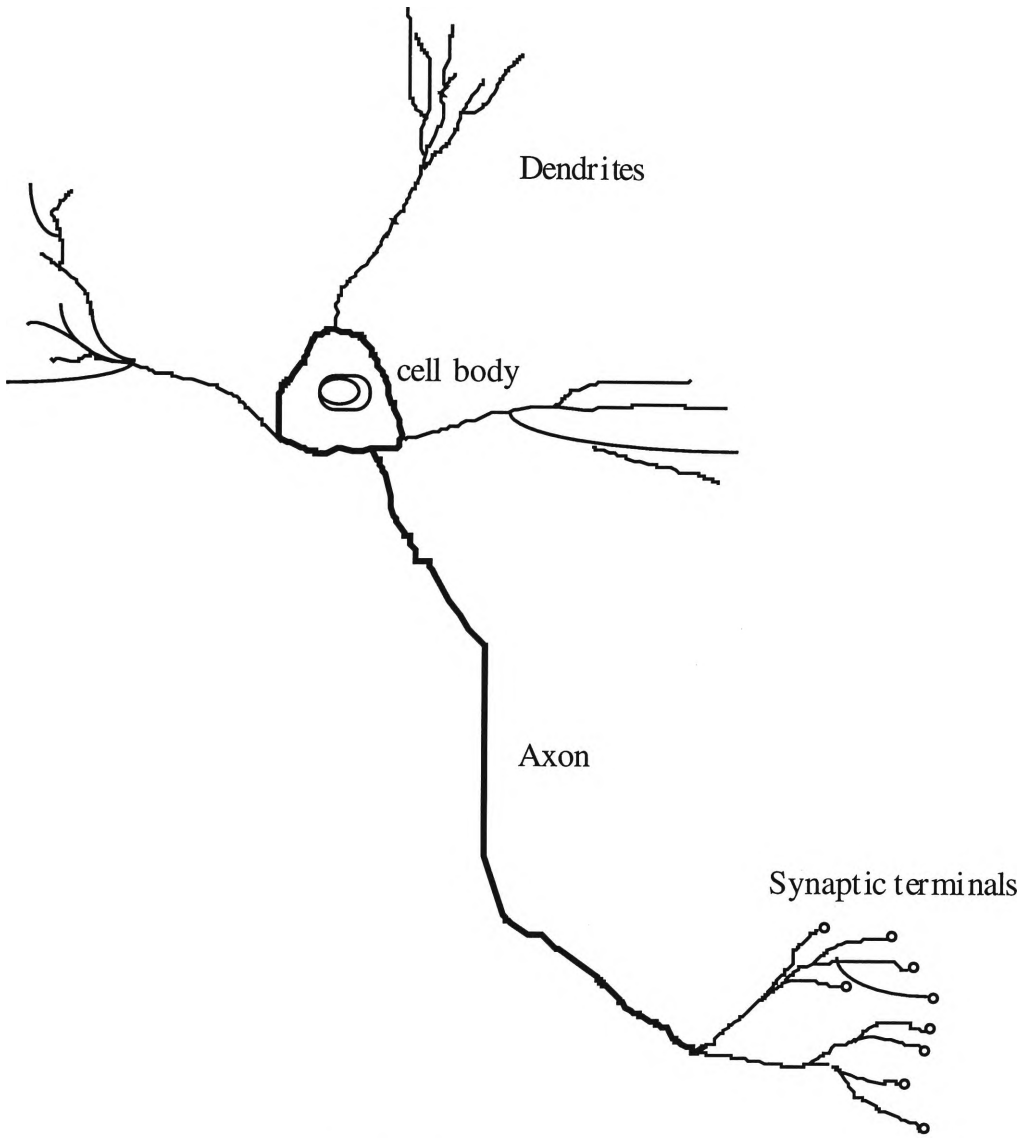


Fig. 3.1 Structure of a neuron

The interactions between neurons are mediated through synapses. The most common kind of synapse is a chemical synapse, which operates as follows. A presynaptic process liberates a transmitting substance that diffuses across the synaptic junction between neurons and then acts on a post-synaptic process. Thus a synapse converts a presynaptic electrical signal into a chemical signal and then back into a postsynaptic electrical signal or stimulus (Shepherd and Koch, 1990). It is rather like a nonreciprocal two-port device. Through this way, information is transmitted through nerve system.

A neural network is an artificial mathematical model, which tries to imitate the way that the brain works. They are composed of a number of nodes, or units, connected by links. Each connection has a numeric weight associated with it, and the weight is updated throughout the learning process. Some of the nodes connected to the external environment can be designated as the input or output of the network.

The assignment of a neural network to perform a certain task is carried out through several steps. One must decide how many nodes are to be used, what are the input, and output, what kind of node is appropriate for a certain unit, and how the nodes are to be interconnected. Then, the network is trained after initialization, with all the weights updating during training. There may be several learning algorithms to be employed.

To achieve good performance, neural networks employ a massive interconnection of simple computing cells. From the view of adaptive machine, the following definition may be adopted (Aleksander and Morton, 1990):

A neural network is a massively parallel distributed processor that has a natural propensity for storing experiential knowledge and making it available for use. It resembles the brain in two respects:
(i) Knowledge is acquired by the network through a learning process.
(ii) Interneuron connection strengths known as synaptic weights are used to store the knowledge.

In the learning process (learning algorithm) the synaptic weights are modified of the network in such a fashion as to attain a desired objective.

3.1.2 Simple computing elements (Neuron Model)

The neural networks are made up of massive simple elements, or nodes. Each node performs a simple computation; it receives signal from its input connections and computes a new activation level that it sends along to each of its output connections. Usually the element is divided into parts and shown in Fig. 3.2.

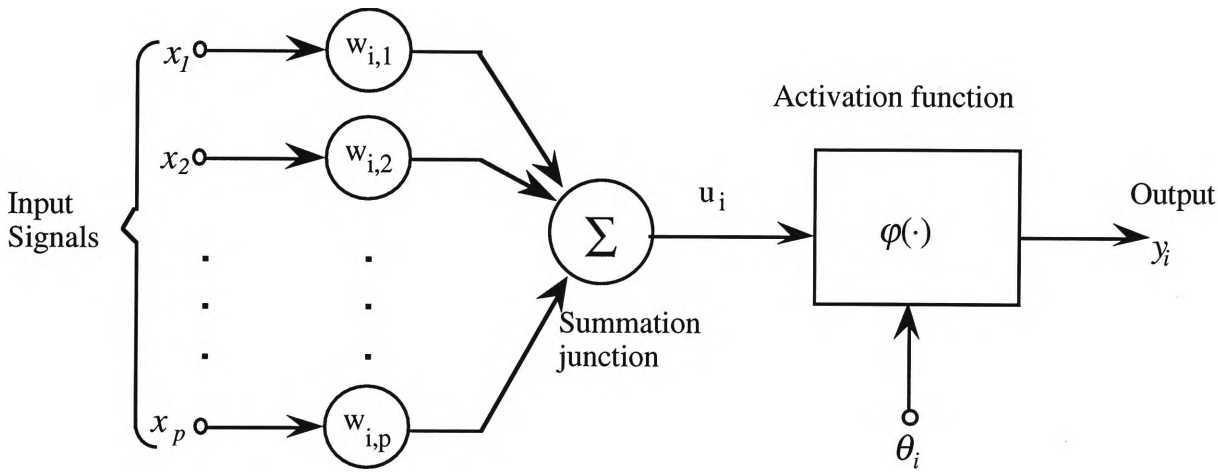


Fig. 3.2 The model of a nonlinear neuron

The simple computing element is known as a neuron. A neuron generally has two parts:

1. Input function.

There are a set of synapses or connecting links connected to a neuron, each of which is embedded with a weight or strength. Specifically, a signal x_j at the input of link j connected to a neuron i is multiplied by the synaptic weight $w_{i,j}$. The input signals are then summed. The total weighted input is the sum of the input activation times their respective weights:

$$u_i = \sum_{j=1}^{p} w_{i,j} x_j \tag{3-1}$$

where x_1, x_2, ..., x_p are the input signals, $w_{i,1}$, $w_{i,2}$, ..., $w_{i,p}$ are the weights of the links and u_i is the linear output from input function summation.

2. Activation Function.

An activation function is for limiting the amplitude of the output of a neuron. It squashes the permissible amplitude range of the output signal to some finite value. Typically, the normalized amplitude range of the output of a neuron is set as the closed unit interval [0, 1] or alternatively [-1,1].

The model shown in Fig. 3.2 includes an externally applied threshold θ_i that would lower the net input of the activation function. On the other hand, the net input of the activation function may be increased by employing a bias term rather than a threshold. The bias is the negative of the threshold. In the application, there is always bias connected to the neurons. On the presence of θ_i, the activation function is expressed as:

$$y_i = \varphi(u_i - \theta_i) \tag{3-2}$$

where $\varphi()$ is the activation function, and y_i the output signal from the neuron.

This kind of model can be modified in other ways. The effect of θ_i can be included in the input function part. A new input may be added with a synapse weight of θ_i, the input is fixed at -1. Another way to modify the model is to add an input with fixed input x_0=+1 and the weight $w_{i,0}=b_i$, which accounts for bias b_i. The latter one, illustrated in Fig. 3.3, is often adopted in neural networks applications. Mathematically, these models are all equivalent.

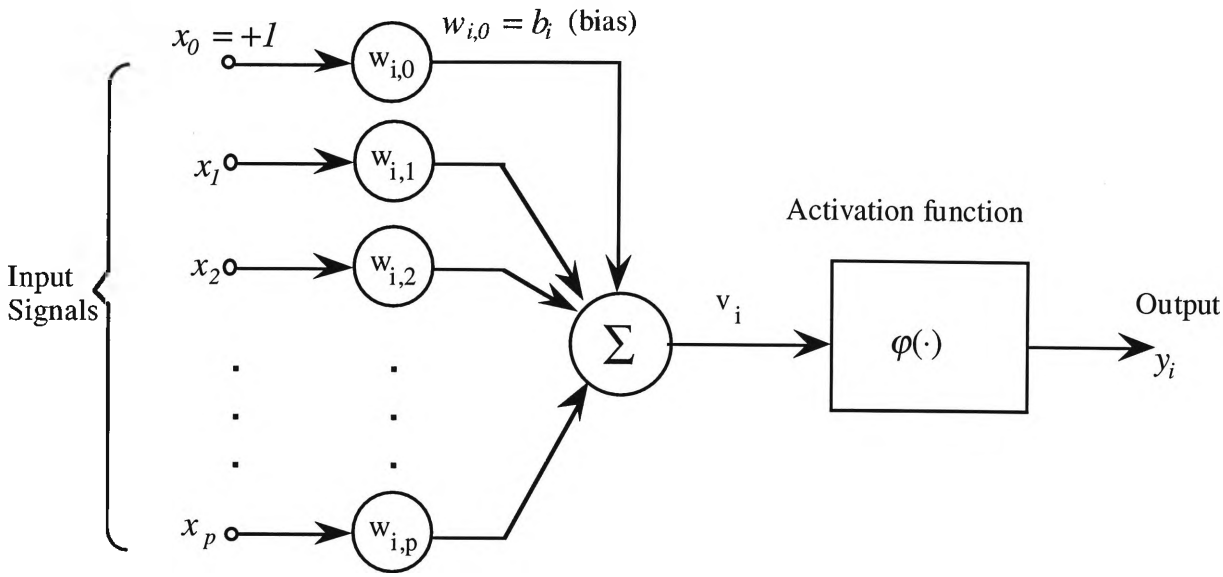


Fig. 3.3 Nonlinear model with bias

There are a large selection of activation functions. Among them, three types are typically noted:

(i) Threshold Function, it is shown in Fig. 3.4

$$\varphi(x) = \begin{cases} 1 & if \quad x \geq t \\ 0 & if \quad x < t \end{cases} \tag{3-3}$$

where $t \subset R$.

(ii) Piecewise Linear Function. A typical piecewise function would be like the one in Fig. 3.5.

(iii) Sigmoid Function. The sigmoid function is the most commonly used form of activation function in Artificial Neural Networks. It is mathematically defined as:

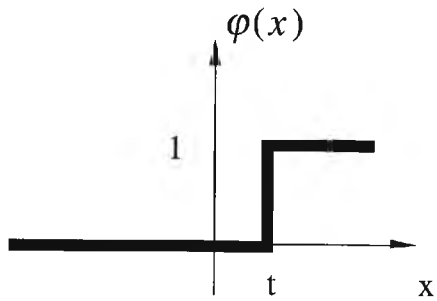


Fig. 3.4 Threshold function

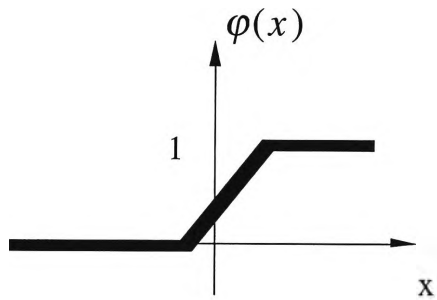


Fig. 3.5 Piecewise function

$$\varphi(x) = \frac{1}{1+e^{-ax}} \tag{3-4}$$

where a is the slope parameter of the sigmoid function. In most cases, a is assigned a value of +1, thus the sigmoid function becomes

$$\varphi(x) = \frac{1}{1+e^{-x}} \tag{3-5}$$

The sigmoid function is illustrated in Fig. 3.6.

A notable thing in application of Neural Networks is that the output of activation function can be in [-1, +1] instead of [0, 1]. This would provide a choice for the construction of neural networks. In such case, a sigmoid function would be replaced by the hyperbolic tangent function,

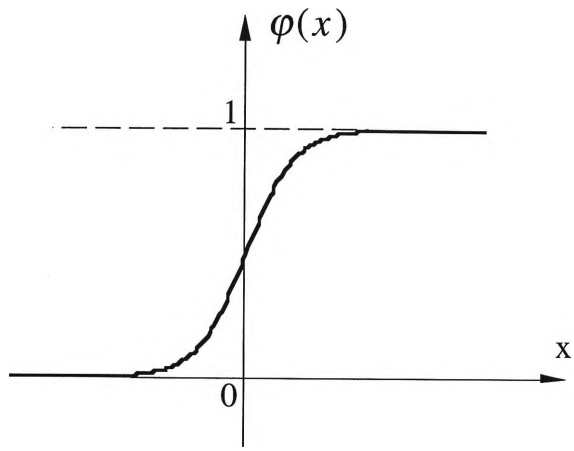


Fig. 3.6 Sigmoid function

$$\varphi(x) = \tanh(x/2) = \frac{1-e^{-x}}{1+e^{-x}} \tag{3-6}$$

3.1.3 Neural Networks Structure

3.1.3.1 Neural Networks graphics representation

Up to now, the basic computing components of a neuron have been briefly introduced. Although it is an accurate description of the model, graphic structure for neuron in Fig. 3.2 is hardly used in a neural network structure. Rather a signal-flow graph is used in representing either a neuron or a neural network. It omits the signal flow inside the individual neurons. It can be characterized as follows:

(i) Source nodes supply input signals to the graph.

(ii) Each neuron is represented by a single node called computation node.

(iii) The communication links interconnecting the source and computation nodes of the graph carry no weight; they merely provide directions of signal flow in the graph.

This kind of signal flow graph will be used to describe the architecture of the neural network. It is illustrated in Fig. 3.7. The convention for this kind of representation is to

use a circle to represent a computing node and a small square to represent a source input. It would be used in describing neural networks in this thesis.

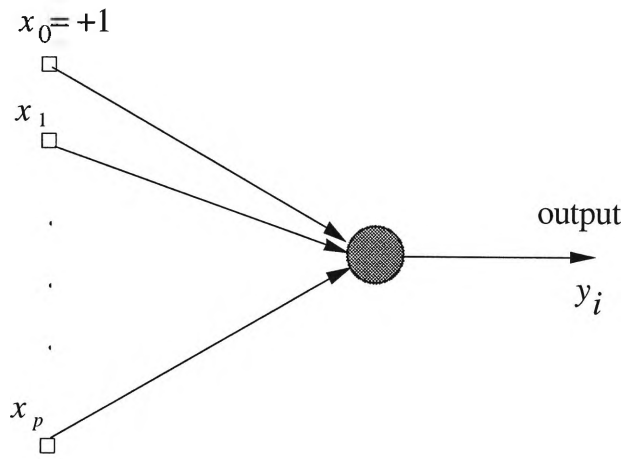


Fig. 3.7 Graphic representation

3.1.3.2 Neural Networks structure classification

There are a number of network structures, each of which result in very different computational properties. In general, two classes of network architectures are identified, the *feed-forward* and *recurrent* networks. In a feed-forward net work, links are unidirectional, and there are no cycles. In a recurrent network, the links can form arbitrary topologies. Technically, a feed-forward network is a directed acyclic graph. Usually when dealing with neural networks like a network of neurons organized in the form of layers. In a layered feed-forward network, each unit is linked only to units in the next layer; there is no links between units in the same layer, no links backward to a previous layer, and no links that skip a layer. Apparently, the brain, with massively inter-connected topology, cannot be a feed-forward network. In the above terminology, the brain is a recurrent network.

Although the feed-forward network cannot be a replica of the brain, it has its own significance in computing. The lack of cycles predetermined that computation can proceed uniformly from input units to output units. The activation from the previous time

step plays no part in the computation. Hence, a feed-forward network simply computes a function depends on the weight setting, as there is no internal state other than the weight; All facts that the neural networks can *learn* is stored in weights.

A layered network is a network of neurons organized in the form of layers. The input layer of source nodes and output layer of output nodes are basic to the neural networks. A feed-forward network with more than two layers is called multilayer feed-forward network. The layers, which are other than input layer and output layer, are called hidden layers. All the nodes in the hidden layer are called hidden neurons. The function of the hidden layers is to interact between the input and output. By adding hidden layers, the network is enabled to extract higher-order statistics. In other words, provides an overall view for the networks.

A fully connected feed-forward network is like the one shown in Fig. 3.8, which has one hidden layer. It is said to be fully connected in that every node in each layer is connected to every node in the adjacent forward layer.

3.2 Neural Networks learning theory and learning algorithms

Learning is the basic feature of a neural network, which emulates the brain function. By learning, it gives the network the ability to *learn* from environment, and to improve its performance.

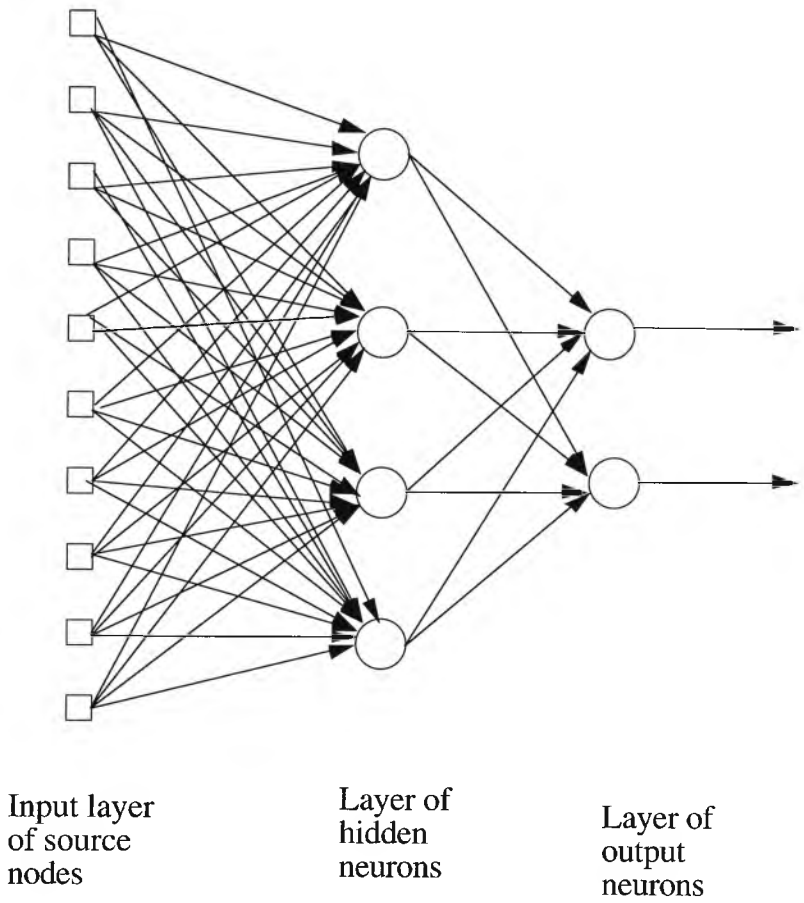


Fig. 3.8 Fully connected feed-forward network

By classification, there are many ways of learning, with each a different learning algorithm. This section focuses more on the Back-propagation neural network, as it is the kind of neural networks used as a modeling tool for this thesis.

Throughout the learning process, the neural network undergoes changes as a result of stimulation from external environment. As all the information of a neural network is stored in the weights that interconnect the neurons, the proper learning algorithm adjusts the synaptic weights towards a desired output under specific stimulus (inputs) for the neural network.

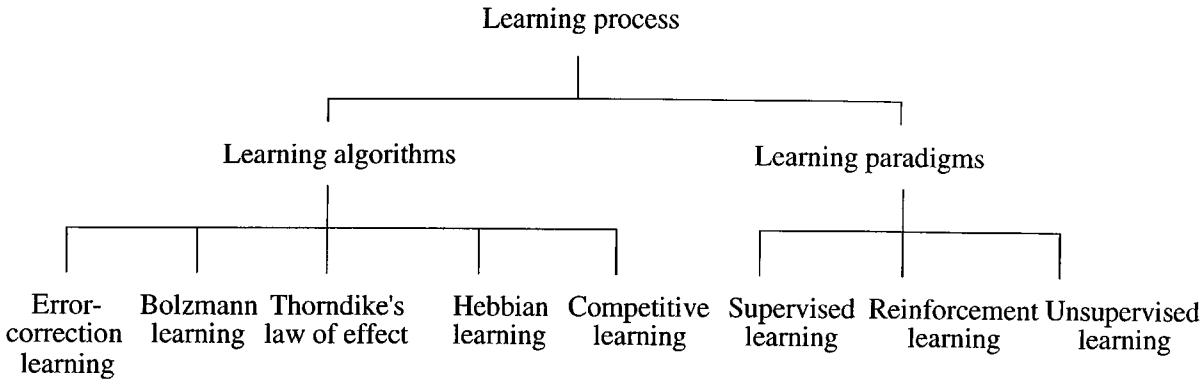


Fig. 3.9 A taxonomy of the learning process

The different division of learning process is classified in Fig. 3.9 (Haykin). Error-correction is rooted in optimum filtering. Both Hebbian learning and competitive learning have neurobiological background. Bolzmann learning borrowed its ideas from thermodynamics and information theory. The three classes of learning paradigms are supervised learning, reinforcement learning, and unsupervised learning. Supervised learning, as its name implies, is performed under the supervision of an external supervisor. Reinforcement learning is the evolving process through trial and error. Unsupervised learning is, unlike the former two, performed in self-organized manner. The back-propagation network is a supervised learning with its learning algorithm based on error-correction.

3.2.1 Error-Correction Learning

Suppose neuron k has a desired response $d_k(n)$ at time n for the input vector $\mathbf{X}(n)$. The actual response is denoted by $y_k(n)$. Usually, the actual response $y_k(n)$ of neuron k is different from the desired response $d_k(n)$. Hence, an error signal $e_k(n)$ is defined by

$$e_k(n) = d_k(n) - y_k(n) \tag{3-7}$$

The ultimate goal of error-correction is to minimize a cost function based on the error signal. A commonly used cost function is the mean-square-error:

$$F = E[\frac{1}{2}\sum e_k^2(n)] \tag{3-8}$$

where E is the statistical expectation, and the summation is over all the neurons in the output layer of the neural network. To minimize the cost function F, the method of gradient descent (Haykin, 1991, Widrow and Stearns, 1985) is used, which is the basis of back-propagation. Practically, it is very difficult to use this equation that requires specific statistical knowledge of the underlying procedure. The solution is to use an approximate substitute.

$$G(n) = \frac{1}{2}\sum_k e_k^2(n) \tag{3-9}$$

It is composed of instantaneous value of the sum of squared errors. The G(n) is minimized through the correction of the synaptic weight of the network, thus, a optimal network. The method of changing weighting, so called error-correction learning rule or delta rule, is to adjust W at time n by (Widrow and Hoff, 1960)

$$\Delta w_{ki}(n) = \eta e_k(n) x_i(n) \tag{3-10}$$

where η is a positive constant, called learning rate. The adjustment of synaptic weight is determined by the error signal, input signal of the synapse and the learning rate. This can be explained in another way. A plot of the cost function F versus the synaptic weights characterizing the neural network consists of a multidimensional surface, an error surface. The learning of the network is trying to find a minimal point of the error surface. When the neural network has nonlinear processing units, it has both global minimum and local minim, both of which can be attained through the learning process. In either case, the network converges. The difference is just the scale of error.

Sometimes, the network does not necessarily converge. This is influenced by all the parameters of the network. The error-correction learning behaves like a feed-back system. The choice of different values of learning rate can mean the difference between

convergence and divergence. Learning rates also determine the speed of convergence of the network.

3.2.2 Supervised Learning

The supervised learning is so called as if an external teacher supervises all the time. The essential ingredient of the teacher can be explained as a set of input-output examples from the environment. The whole relationship can be illustrated in Fig. 3.10. There is an assumed teacher, who knows the ideal input-output examples. The neural networks are then trained under this teacher. The teacher would direct a desired response (output) to the network when there is an input to it. In other words, the teacher and networks are exposed to the input at the same time, but the teacher knows the correct response, while the untrained neural network knows nothing about the output of interest. Then the teacher would make the networks to produce as close as possible the ideal output. The desired response, then, is an optimum action for the network to perform. The network parameters are adjusted under the combined influence of the training vector and the error signal. The adjustment is carried out in a step by step fashion with the aim that eventually, the neural network emulates the teacher. This emulation is presumed to be optimum in some statistical sense or by the virtue of knowledge.

The algorithm of error correction is a kind of supervised learning. The optimum target of a network is to minimize the cost function of interest. It is like a closed-loop feedback system. The performance measure used for a supervised learning system is defined in terms of a set of targets (i.e. desired responses) by means of a known error criterion (e.g., mean square error). This feedback system is viewed as instructive.

Supervised learning can be performed in an off-line or on-line manner. In the off-line case, a separate computational facility is used to design the supervised learning system. Once the desired performance is accomplished, or it is considered acceptable, the design is "frozen", which means all the parameters of the neural network are fixed and the network operates in a static manner. On the other hand, the on-line learning means that the learning is accomplished in real time. The neural network is dynamic. In both cases, the neural network cannot learn without a teacher. If there is a case not covered by examples used to train the network, the neural network would not know the optimum output. Thus the output can be of some casual value. This is a limitation of supervised learning.

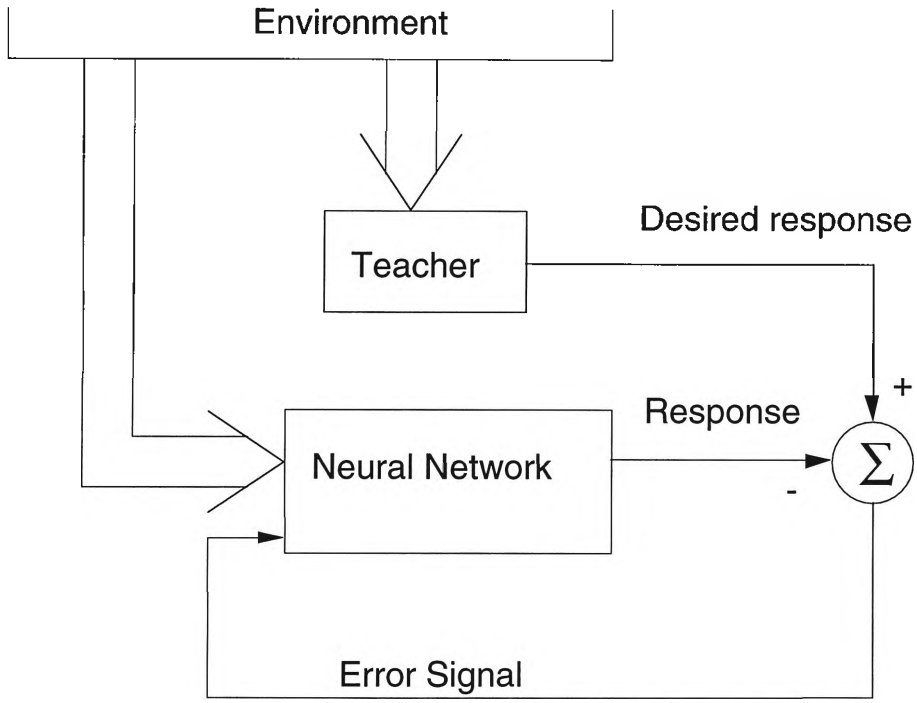


Fig. 3.10 Supervised learning: block design

Different from supervised learning, reinforcement learning is the on-line learning of an input-output mapping through a process of trial and error designed to maximize a scalar

performance index called reinforcement signal. Another learning paradigm is the unsupervised learning or self-organized learning. There is no external teacher or critic to oversee the learning process.

3.3 The Back-Propagation neural network

3.3.1 The perceptron

Layered feed-forward networks were first studied in the late 1950s under the name perceptrons. The perceptron is the simplest form of a neural network used for the classification of a special type of patterns said to be linearly separable(i.e. patterns that lie on opposite sides of a hyperplane). The perceptron is also the basic element of the multi-layered perceptrons, which is actually the most widely used Back-Propagation.

Frank Rosenblatt (1958, 1962) first proved that a learning system using the perceptron learning rule will converge to a set of weights that correctly represents the examples, as long as the examples represent a linearly separable function. The proof of the algorithm is known as the perceptron convergence theorem. Fig. 3.11 depicts a single perceptron and its signal-flow graph.

The linear summation of all input signals is

$$v_i = \sum_{i=1}^{p} w_i x_i - \theta` \tag{3-11}$$

The purpose of the perceptron is to classify the set of externally applied stimuli $x_1, x_2, ..., x_p$ into one of two classes, 1 or 2. The decision rule for the classification is to assign the point represented by the inputs $x_1, x_2, ..., x_p$ to class 1 if the perceptron output y is +1 and to class 2 if it is -1. The decision boundary is defined by

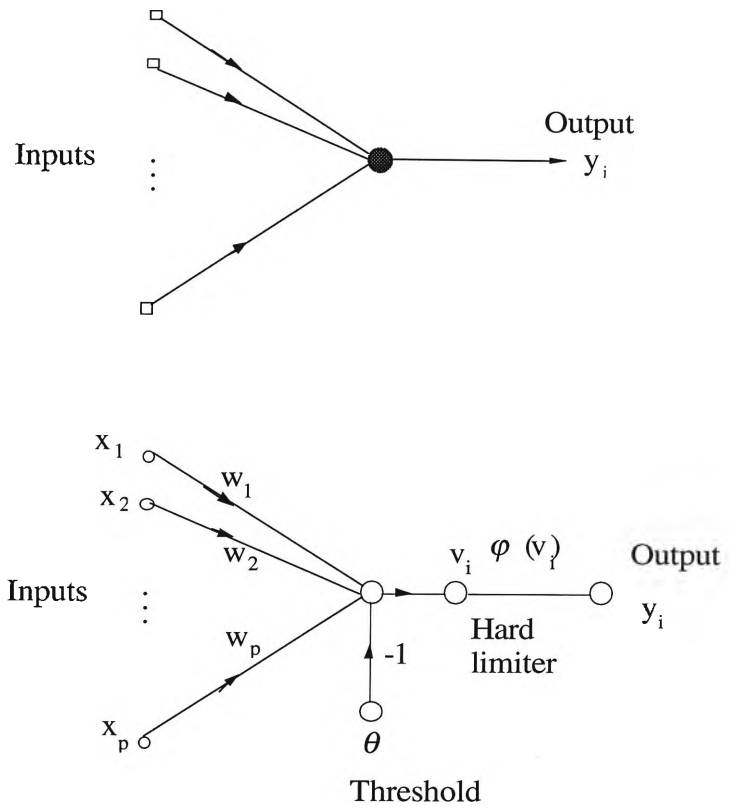


Fig. 3.11 Single perceptron and its signal flow

$$\sum_{i=1}^{p} w_i x_i - \theta = 0 \tag{3-12}$$

It is shown in Fig. 3.12 for the case of two input variables x_1 and x_2. The perceptron use the error-correction rule to adjust its weight vector.

The weight vector adjustment can be defined using error-correction described before.

$$\Delta \mathbf{w}(n) = -\eta \nabla_{\mathbf{w}} \hat{F}(n) = \eta[d(n) - y(n)]\mathbf{x}(n) \tag{3-13}$$

where η is the learning rate, d(n), the desired response, y(n), the actual response and x(n), the input vector (Shynk, 1990)

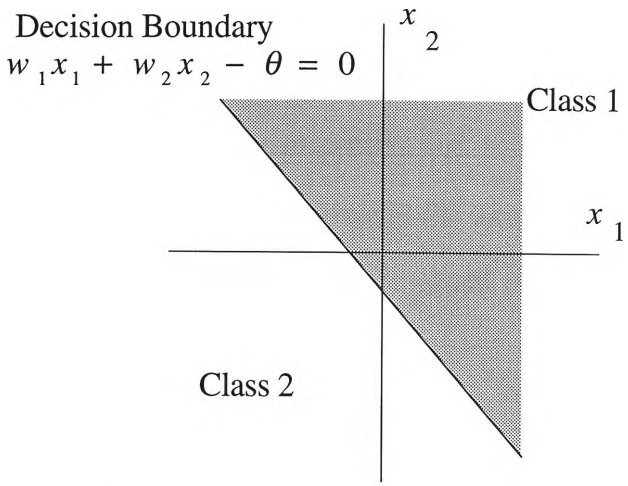


Fig. 3.12 Two variables decision boundary

3.3.2 The Multilayer Perceptrons

The multilayer perceptrons, namely, is a multilayer feed-forward network. The network consists of a set of sensory units (source nodes) that constitute the input layer, one or more hidden layers of computation nodes, and an output layer of computation nodes. The algorithm used by multilayer perceptrons, or the back-propagation algorithm, is based on the error-correction learning rule. It has an algorithm much like least-mean-square algorithm. (The least-mean-square (LMS) algorithm, is also known as the delta rule or the Wildrow-Hoff rule (Widrow and Hoff, 1960). The LMS algorithm was originally formulated by Widrow and Hoff for use in adaptive switch circuits. The network was called adaline, which was inspired by Rosenblatt's perceptron.)

Basically, the error back-propagation process consists of two passes through the different layers of the network: a *forward* pass and a *backward* pass. In the forward pass, an activity pattern (input vector) is applied to the sensory nodes of the network, and its effect propagates through the network, layer by layer. Finally, a set of outputs is produced as the actual response of the network. During the forward pass the synaptic weights of the network are all fixed. On the other hand, during the backward pass, the

synaptic weights are all adjusted in accordance with the error-correction rule. Specifically, the actual response of the network is subtracted from a desired response to produce an error signal. Then, this error signal is then propagated backward through the network, against the direction of synaptic connections-hence the name " error back-propagation." The synaptic weights are adjusted so as to make the actual response of the network move closer to the desired response.

The back-propagation was first described by Werbos in his Ph.D thesis (Werbos, 1974) the context of general networks with neural networks representing a special case. It was rediscovered after 1985 (Rumelhart, Hinton, and Williams, 1986) and was evolving since then.

Some preliminary information will be given before starting describing the algorithm in detail. In general, the network that we are studying is fully connected, which means that a neuron in any layer of the network is connected to all the nodes/neurons in the previous layer. Fig. 3.13 depicts a portion of the multilayer perceptron. There are two kinds of signals identified. (Parker, 1987):

(i). Function Signals. A function signal is an input signal (stimulus) that comes in at the input end of the network, propagates forward (neuron by neuron) through the network, and emerges at the output end of the network as an output signal.

(ii). Error Signals. An error signal originates at an output neuron of the network, and propagates backward (layer by layer) through the network.

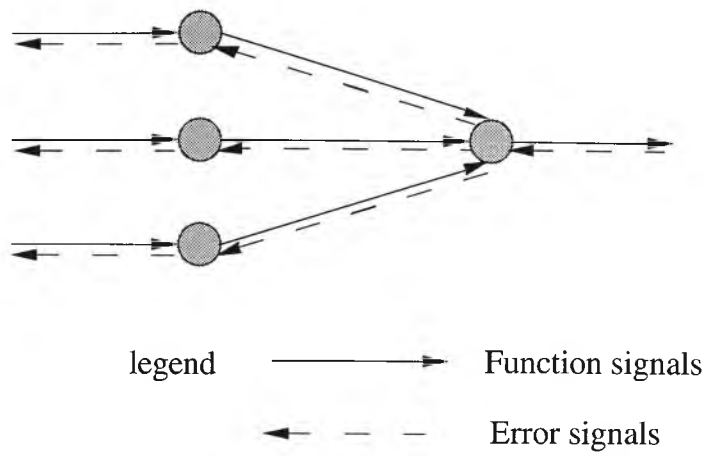


Fig. 3.13 Illustration of the two basic signal flow

Derivation of the Back-Propagation Algorithm

The error signal at the output neuron i at iteration n (i.e. at the nth training pattern) is defined by

$$e_i(n) = d_i(n) - y_i(n) \tag{3-14}$$

The instantaneous value G(n) of the sum of squared errors is obtained by summing $\frac{1}{2}e_i^2(n)$ over all neurons in the output layer.

$$\text{G(n)} = \frac{1}{2}\sum_{i \in X} e_i^2(n) \tag{3-15}$$

where the set X includes all the neurons in the output layer of the network. Let N denote the total number of patterns (examples) contained in the trained set. The average squared error is obtained as

$$G_{AVER} = \frac{1}{N}\sum_{n=1}^{N} G(n) \tag{3-16}$$

For a given training set, G_{AVER} represents the cost function as the measure of training set learning performance. The objective of the learning process is to adjust the free parameters of the network so as to minimize G_{AVER}. To do this, we use an approximation

similar in rationale to that of the LMS algorithm. Since the network is trained by a set of patterns, the weights are updated on a pattern by pattern basis.

Consider a neuron I being fed by a set of function signals produced by a layer of neurons to its left. The net internal activity level $v_i(n)$ produced at the input of nonlinearity associated with neuron i is therefore

$$v_i(n) = \sum_{j=0}^{p} w_{ij}(n) y_j(n) \tag{3-17}$$

where p is the total number of inputs (excluding the threshold) applied to neuron i. The synaptic weight w_{i0} (corresponding to the fixed input y_0=-1) equals the threshold θ_i applied to neuron i. Hence the function signal appearing at the output of neuron i at iteration n is

$$y_i(n) = \varphi_i(v_i(n)) \tag{3-18}$$

The back propagation algorithm needs to apply a correction to the synaptic weight, which is proportional to the instantaneous gradient $\partial G(n)/\partial w_{ij}(n)$. The gradient may be expressed as

$$\frac{\partial G(n)}{\partial w_{ij}(n)} = \frac{\partial G(n)\partial e_i(n)\partial y_i(n)\partial v_i(n)}{\partial e_i(n)\partial y_i(n)\partial v_i(n)\partial w_{ij}(n)} \tag{3-19}$$

This gradient determines the direction of search in weight space for the synaptic weight $w_{ij}(n)$. After applying the differentials from Eq.(3-14) (3-15) (3-18) , we get

$$\frac{\partial G(n)}{\partial w_{ij}(n)} = -e_i(n)\varphi_i'(v_i(n))y_i(n) \tag{3-20}$$

The correction $\Delta w_{ij}(n)$ applied to $w_{ij}(n)$ is defined by the delta rule

$$\Delta w_{ij}(n) = -\eta \frac{\partial G(n)}{\partial w_{ij}(n)} \tag{3-21}$$

where η is a constant that determines the rate of learning; it is called learning-rate parameter of the back propagation algorithm. The use of minus sign in Eq.(3-21) accounts for gradient descent in weight space. Thus, we may obtain

$$\Delta w_{ij}(n) = \eta \delta_i(n) y_i(n) \tag{3-22}$$

where the local gradient $\delta_i(n)$ is defined by

$$\delta_i(n) = -\frac{\partial G(n) \partial e_i(n) \partial y_i(n)}{\partial e_i(n) \partial y_i(n) \partial v_i(n)}$$

$$= e_i(n) \varphi_i'(v_i(n)) \tag{3-23}$$

The local gradient for output neuron i is equal to the product of the corresponding error signal and the derivative of the associated activation function. It points to required changes in synaptic weights.

The above discussed weight adjustment is for output neurons. For neurons in hidden layers, the back-propagation algorithm also makes adjustment for synaptic weights for neurons to its previous layer. It back-propagates the error signals through the network.

When neuron i is located in a hidden layer of the network, there is no specified desired response for that neuron. Accordingly, the error signal for a hidden neuron would have to be determined recursively in terms of the error signals of the neurons to which that hidden neuron is directly connected. The local gradient for hidden layer is described as

$$\delta_i(n) = -\frac{\partial G(n \partial y_i(n)}{\partial y_i(n) \partial v_i(n)}$$

$$= -\frac{\partial G(n)}{\partial y_i(n)} \varphi_i'(v_i(n)), \qquad \text{neuron i is hidden} \tag{3-24}$$

while the error signal is $G(n) = \frac{1}{2} \sum_{k \in X} e_k^2(n)$, neuron k is an output node.

The partial derivative can be obtained as

$$\frac{\partial G(n)}{\partial y_i(n)} = -\sum_k \delta_k(n) w_{ki}(n) \tag{3-25}$$

Using Eq.(3-24), the local gradient for hidden layer $\delta_i(n)$ is determined as follows,

$$\delta_i(n) = \varphi_i'(v_i(n))\sum_k \delta_k(n) w_{ki}(n) \tag{3-26}$$

This equation also applies to hidden layer with more hidden layers to its right. The local gradient depends on error signals for all those neurons that lie in the layer to the immediate right of hidden neuron i, and the synaptic weights associated with these connections. The synaptic weights for neuron i and its input signals can then be adjusted according to the local gradient, using Eq.(3-26).

In the application of back-propagation algorithm, two distinct stages of computation may be distinguished. The first stage is referred to as the forward pass, and the second one as the backward pass.

In the forward pass, all the weights remain unaltered. The function of signals are computed on a neuron by neuron, and layer by layer basis. With input signals (input vector) from input layer, the network transfer signals through the hidden layers to the output layer. At the end of the output layer, the output vector is generated, which is compared to the desired output signal. The error between the two vectors (actual response and desired response) is measured by using a cost function.

The backward pass, on the other hand, starts at the output layer by passing the error signals backward through the network, layer by layer. It recursively computes the local gradient $\delta_i(n)$ for each neuron. Under such recursive process, the synaptic weights throughout the neuron would be updated in accordance with the delta rule layer by layer. The goal of this change is to minimise the cost function. Through this discussion, we can easily understand the reason of the name of Back-Propagation.

3.3.3 The convergence of BP

There are many factors affecting the convergence and speed of convergence of the Back-Propagation network. Also, concerning the convergence, some modification of the algorithm is added. This is discussed in further detail below.

(i) Momentum

The back-propagation algorithm provides an approximation to the trajectory in weight space computed by the method of steepest descent. The smaller we make the learning rate parameter η , the smaller the changes to the synaptic weights in the network will be from one iteration to the next and the smoother will be the trajectory in weight space. The cost is that the rate of learning is very slow. However, if the η is too large, it also creates problems like unstable network and the network may not converge. A modification of this algorithm would involve the introduction of a momentum term, as shown by (Rumelhart et al. 1986a)

$$\Delta w_{ij}(n) = \alpha \Delta w_{ij}(n-1) + \eta \delta_i(n) y_i(n) \tag{3-27}$$

where α is usually a positive number called the momentum constant. This is called generalize delta rule and delta rule is just a special case when α=0. The incorporation of momentum in the back-propagation algorithm would highly benefit the learning behavior of the algorithm and the stability of the network. It also has the effect of preventing the learning process from terminating in a shallow local minimum on the error surface.

(ii) Batch mode training.

In a practical application of the back-propagation algorithm, learning results from the many presentations of a prescribed set of training examples to the multilayer perceptron. One complete presentation of the entire training set during the learning process is called an epoch. The training set is in a randomized order. The learning process is maintained on an epoch by epoch basis until the synaptic weights and threshold level of the network

stabilize and the average squared error over the entire training set converges to some minimal value. For a particular epoch, the cost function is defined as the average square error .

$$G_{AVER} = \frac{1}{2N}\sum_{n=1}^{N}\sum_{i \in X} e_i^2(n) \tag{3-28}$$

In the batch mode the weight adjustment is made only after the whole epoch has been presented to the network. The use of batch mode of training provides a more accurate estimate of the gradient vector.

(iii) Convergence

The back-propagation algorithm is a first-order approximation of the steepest-descent technique in the sense that it depends on the gradient of the instantaneous error surface in weight space. The algorithm is therefore stochastic in nature. The learning is a slow convergence. There are at two fundamental causes for this property (Jacobs, 1988). First, the error surface is fairly flat along a weight dimension, which means the derivative of the error surface with respect to that weight is small in magnitude. Second, the direction of the negative gradient vector may point away from the minimum of the error surface.

When the network converges, there is the problem of local minima. The Back-Propagation is basically a gradient descending (or so called hill-climbing) technique, it runs the risk of being trapped in a local minimum. In other words, the local minima are minima on the error surface which is not deep as the global minimum.

In addressing the problems of convergence, we not only expect the network to stabilize (converge), but also want the network to arrive at a global minimum, where optimal results can be obtained.

3.4 Some consideration of the back-propagation

(i) Initialization

The first step of back-propagation learning is to initialize a network. All the synaptic weights are given initial values. It is the start position for the next work on the error surface. A good assignment of initial values can be of a tremendous help in network design. It will shorten the trace of learning for the network to come to a minimal point on the error surface.

The wrong choice of initial weights can lead to the premature saturation of the network. When a training pattern is applied to the input layer of a multilayer perceptron, the output values of the network are calculated through a sequence of forward computations that involves inner products and sigmoidal transformations. This is followed by a sequence of backward computations that involves the calculation of error signals and pertinent slope of the sigmoidal activation function, and culminates in synaptic weight adjustments. Suppose that, for a particular training pattern, the net internal activity level of an output neuron is computed to have a large magnitude. Hence the corresponding slope of the activation function for the neuron will be very small, and the output value for the neuron will be close to the limiting values (i.e. 0 or +1, if the limiting values are 0 and +1). In such case, the neuron is termed in saturation. When neurons come to saturate, the network will move very slow or stop on the error surface.

In practical application of neural network designs, it adopts two ways of initialization. The first one is to set the free parameters of the network to random numbers that are uniformly distributed inside a small range of values. Alternatively, the free parameters can be set to random numbers that are Gaussian distributed to a certain range. There is no particular preference. The selection of initialization is determined by its performance on the network.

(ii) Other concerns

Another theoretical concern would be the selection of different activation function and network parameters. When the sigmoidal activation function built into the neuron model of the network is symmetric, a multilayer perceptron trained with the back-propagation algorithm may, in general, learn faster (in terms of the number of training iterations required) than when it is nonsymmetric. This alternative function is mentioned in 3.1.2 as the hyperbolic tangent.

All the neurons in the multilayer perceptron should desirably learn at the same rate. Typically, the last layers tend to have larger local gradients than the layers at the front end of the network. Hence, the learning rate parameter η should be assigned a smaller value in the last layers than the front-end layers.

The back-propagation learning is taken as a kind of generalization or approximation. It may be viewed as a curve fitting problem, which is similar to statistical regression. Such a viewpoint allows generalization to be not as a mystical property of neural networks but rather simply as the effect of a good nonlinear interpolation of the input data (Wieland and Leighton, 1987). The network performs useful interpolation primarily because multilayer perceptrons with continuous activation functions leading to output functions that are also continuous.

A neural network that is designed to generalize well will produce a correct input-output mapping even when the input is slightly different from the examples used to train the network. However, if a neural network learns too many specific input-output relations, it is overtrained and tend to confine to the training data. Therefore, it is less able to generalize between similar input-output patterns. This is affected not only by the number

of iteration to train the network, but also by the number of nodes in the hidden layer. It is common that more nodes than necessary are set to generate nonlinear interpolation. To acquire a good network, all these factors need to taken into consideration.

Chapter 4

Data Logging and Processing

The neural network may work well as a modelling tool, but it is based on the premise of it is that it has a valid source of data for training the network. This valid source, as mentioned, is the signals from the Roughing Mill. However, the raw signals are not qualified for training and need to be properly processed. The data logging of the signals from the Roughing Mill and their processing are described in this chapter.

4.1 Data logging from Roughing Mill

As mentioned in chapter 1, there are sensors mounted on the Roughing Mill measuring the variation of the dynamic signals. The quantities are detected and transmitted to the low level controllers, PLCs, through the local area network. The PLCs in use at BHP Steel SPPD HSM are GEM 80 arrays. These PLCs have opto-electro isolated output, which would not affect the normal control of the Roughing Mill. These output are analog test points with certain offsets and scales. All the offsets and scales can be adjusted through programming terminals for the PLCs.

The primary task is to obtain all the signals from the PLCs. This procedure is shown in Fig. 4.1. There are hardware and software programs concerning the data logging.

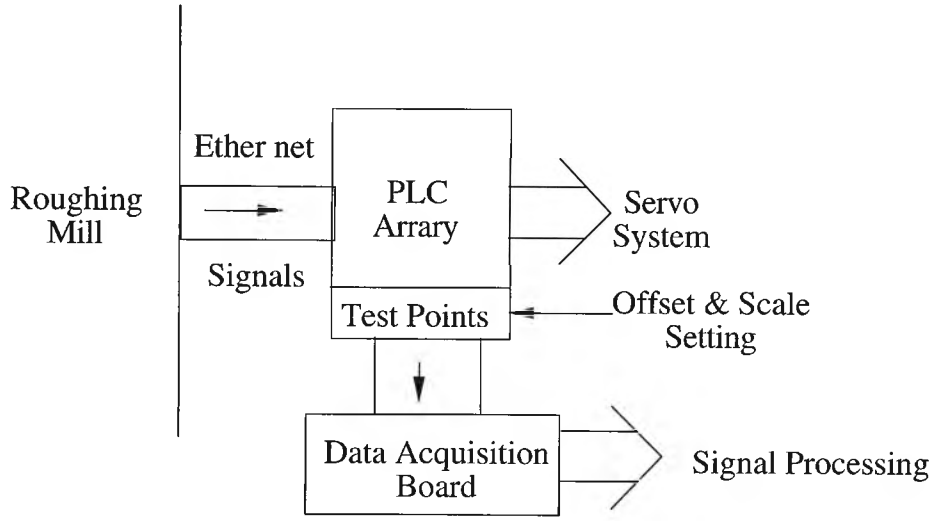


Fig. 4.1 Data logging system

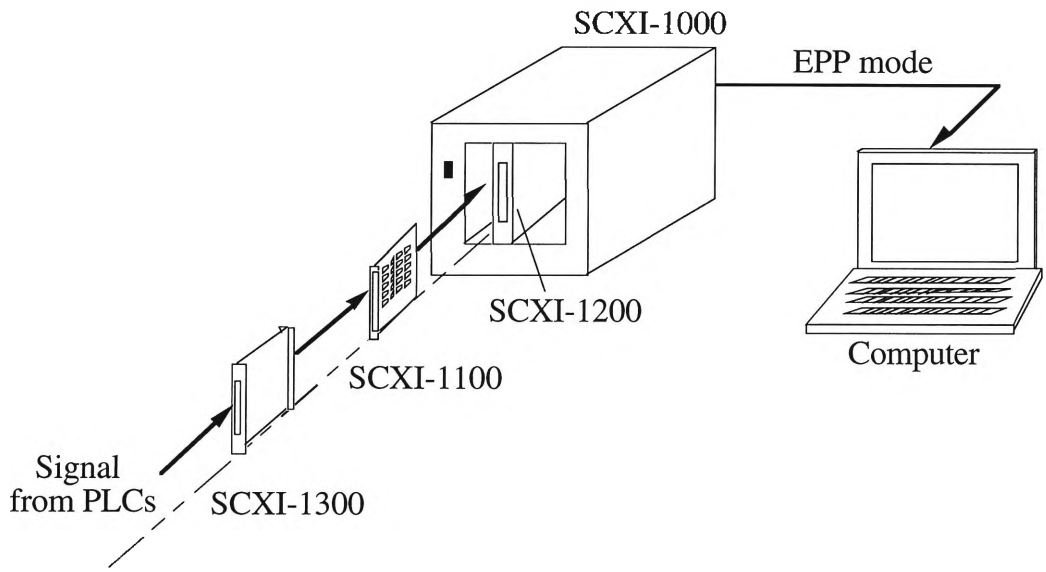


Fig. 4.2 SCXI system

4.1.1 Hardware for data logging

The instruments used for data logging are selected from National Instruments data acquisition products. The system is shown in Fig. 4.2. The computer used is Compaq laptop Concerto, with Intel 80486 DX 33 processor, 12 MB memory and 220 MB hard disk and an enhanced parallel port. SCXI system are optimized for low-level analog signal conditioning, amplification, and filtering.

As shown in Fig. 4.3, analog signals from PLCs output are transferred to the SCXI conditioning modules for quick and easy reconfiguration. The data acquisition board is SCXI-1200, which uses a parallel port connections to the PC. The SCXI-1200 is SCXI module that operates like multifunction analog, digital and timing I/O plug-in board. It

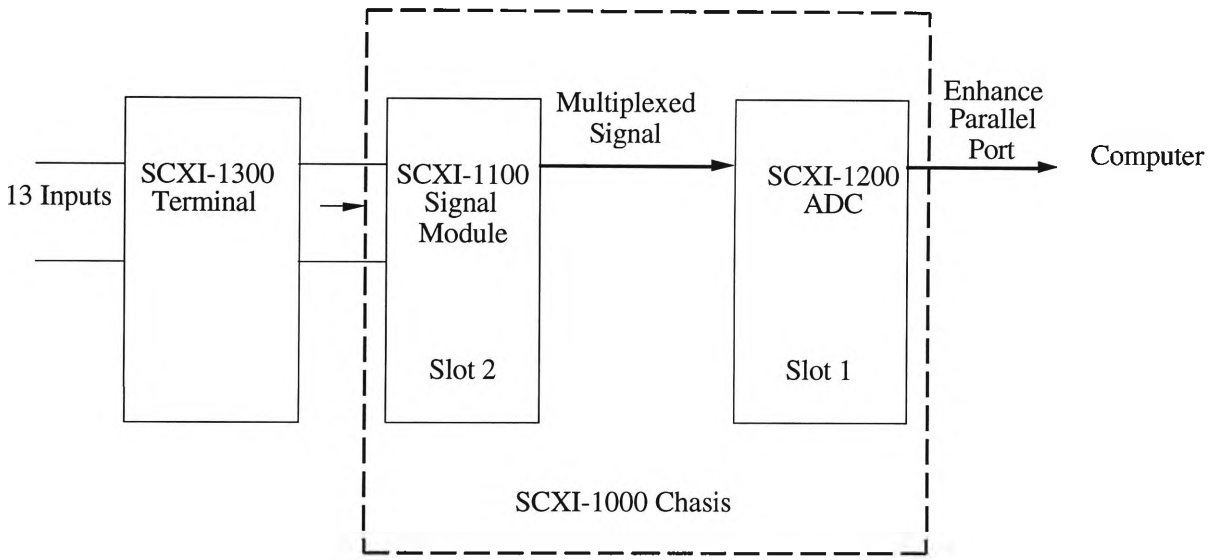


Fig. 4.3 Signal flow in SCXI system

work in a SCXI-1000 four-slot chassis. The SCXI-1200 is compatible with the IEEE 1284 EPP parallel port standard. In Enhanced Parallel Port (EPP) mode, data is transferred to the PC at the rates up to 100 kHz. It has a 12-bit ADC with eight analog

inputs, configurable as eight single-ended or four differential inputs. Since there is a need to sample at least 13 channels of signals, the number of channels available at SCXI-1200 is not enough. Therefore SCXI-1000 and SCXI-1300 have to be used.

The SCXI-1100 operates as a fast 32-channel differential multiplexer with an onboard programmable gain instrumentation amplifier (PGIA). It multiplexes its 32 input channels into a single channel of the data acquisition board. The SCXI-1300 is a general-purpose SCXI terminal block to be used with SCXI-1100 module. It has 64 screw terminals (32 differential channels) for signal connections. Thirteen channels are used for data logging. The signals are multiplexed and transferred through channel 1 of SCXI-1200.

4.1.2 Data acquisition programming

The data acquisition is completed by a software, LabVIEW for Windows.

The Microsoft Windows has become enormously popular because it delivers the ease of graphical computing not possible in character-based DOS systems. Windows offers intuitive graphical user interfaces (GUIs), access to more memory, multitasking, and interprocess communication. The Windows graphical environment is ideal for LabVIEW graphical programming. The Windows graphics engine makes front panel user interfaces and block diagram programs by drawing a pictures. In addition, the multiwindow environment makes it easy to build and access hierarchical VI programs.

LabVIEW, a graphical alternative to conventional programming, is designed for instrumentation and is equipped with the tools you need for test and measurement applications. In LabVIEW, programs can be built which are called virtual instruments (VIs) instead of writing text-based programs. A LabVIEW VI consists of a front panel, a

block diagram, and an icon/connector. The front panel is the user interface, the block diagram is the VI source code, and the icon/connector is the calling interface. A block diagram contains input/output (I/O), computational, and sub-VI components, which are represented by icons and interconnected by lines directing the flow of data. I/O components communicated directly with configured data acquisition boards. Computational components perform arithmetic and other operations. Sub-VI components call other VIs, passing data through their icon/connectors.

The front panel of our program for data logging is shown as Fig. 4.4.

- *Inputting Channels* is the address of incoming signals on terminal SCXI-1300. The address in our setting is, *ob0!sc1!md2!0:12.*

- *Scan Rate* is the frequency used for sampling. It refers to speed per channel. 100 Hz sampling rate was used.

- *Number of Scan Read* is the number of scan samples read from buffer each time. It is set at 200 here.

- *Buffer Sizes* is the size of buffer for each channel (counted by samples). It is set as 3000 samples per channel here.

- *Scan Retrieved* is the number of scans returned. This will be the same as *Number of Scans Read* unless an error or time-out occurs or the VI reaches the end of the data.

- *Scan* Backlog is the amount of data acquired minus the amount of data read.

- Actual *Scan Period* is the time between scans, the inverse of the actual scan rate the VI use to acquire the data. This may differ slightly from the requested scan rate, depending on the hardware capabilities.

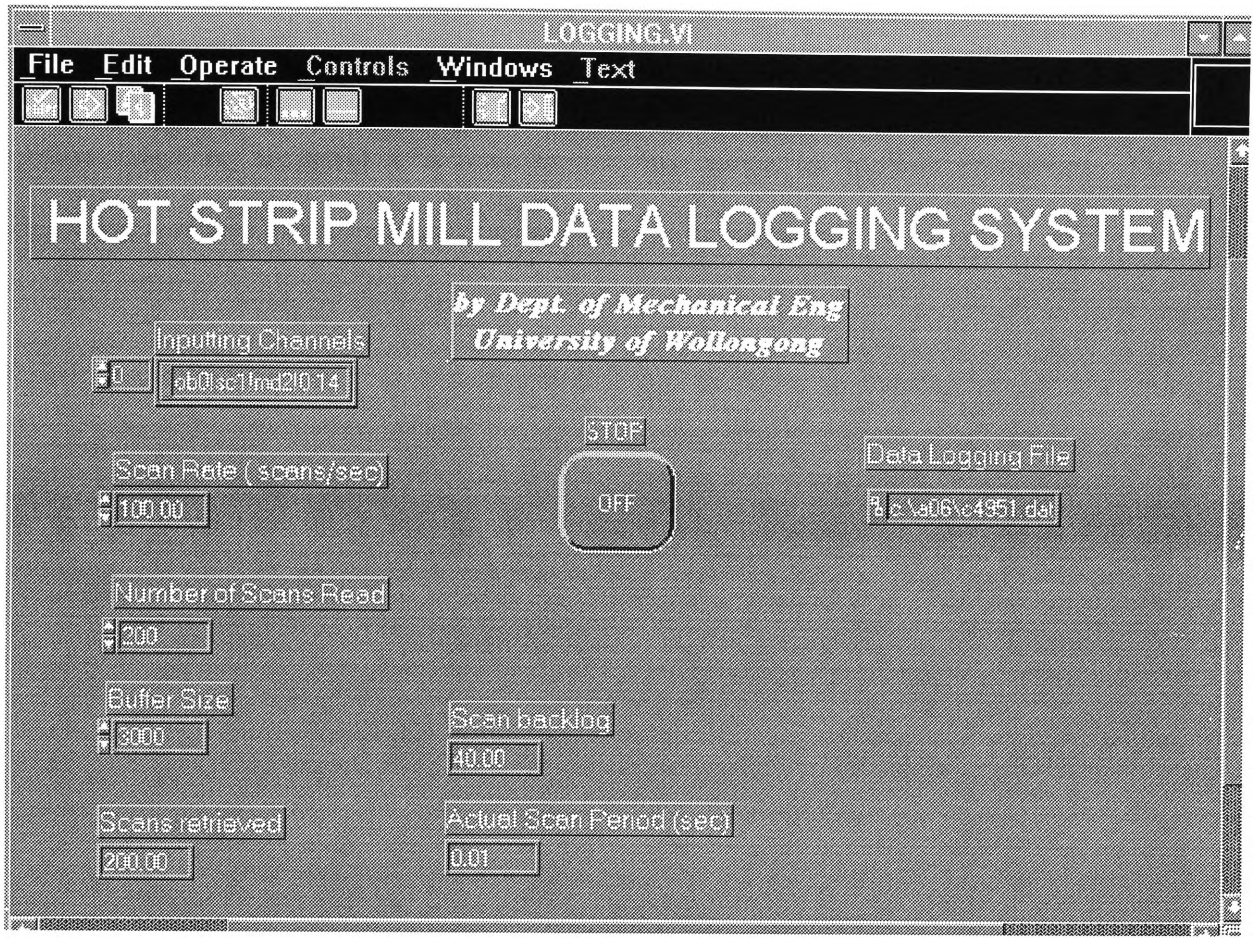


Fig. 4.4 Front Panel of LabVIEW logging program

- *Data Logging File* is the file name used to save the data logged. It includes the complete path for the file.
- The *STOP* icon is used to stop the data logging at any time.

Fig. 4.5 is the block diagram for the logging. The block diagram actually shows the logic relation for each step of logging, the same as common computing language such as C.[1] With this LabVIEW program, signals can be logged from the PLCs and saved on the hard disk for processing.

[1] For full understanding of the program, please refer to , LabVIEW for Windows User Manual and Data Acquisition VI Reference Manual, from National Instrument, 1993

The sampling rate is set to 100 Hz according to the sampling theorem, which requires that an analog signal be sampled at a rate greater than twice its highest frequency. The frequency used would ensure accurate information to be recorded.

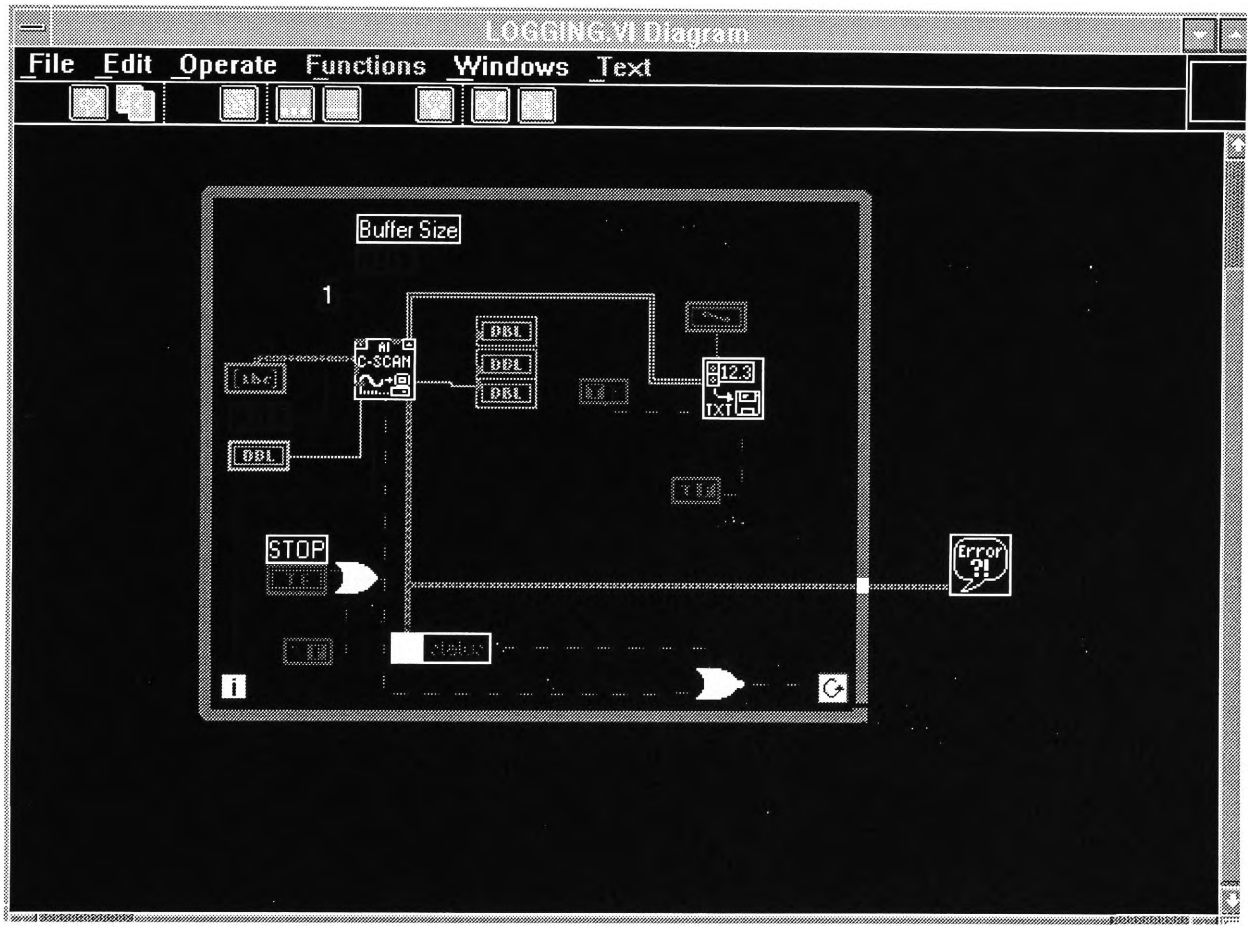


Fig. 4.5 Block diagram of data logging program

4.2 Signal Processing

The signals collected from roughing mill can not be used for neural network model training directly. The signals are not synchronized yet, with noise and outliers in it. A raw signal would not be suitable for modelling unless it has been properly processed. In this

section, we are going to discuss about these proper procedures for signal processing in digital signal domain.

4.2.1 Digital Filtering

The primary purpose of digital filtering is to alter the spectral information contained in an input signal, thus producing an enhanced output signal. This can be done in either time or frequency domain. The program that we used is in time domain. There are four kinds of filters, low-pass filter, high-pass filter, bandpass filter and bandstop filter. Since the high-frequency noise is of our concern, the low-pass filter is chosen.

The program is listed in Appendix C. It is an Lth-order direct form FIR digital filter. The transfer function is described as

$$H(z) = B(z) = b_0 + b_1 z^{-1} + ... + b_L z^{-L} \tag{4-1}$$

which corresponds to the time domain,

$$y_k = \sum_{n=0}^{L} b_n x_{k-n} \tag{4-2}$$

This equation describes an FIR filter, since for an input sequence consisting of a unit impulse at k=0, the filter output y_k is equal to the filter coefficient b_k for k=0,1,..., L and is equal to zero thereafter.

In order to use the FIR filter, we also have to design b_k to fulfill the low-pass filter. The design is also by a program in Appendix C. Only the size of filter, the cutoff frequency and the kind of data window to be used in the design, need to be supplied. Once all the b_k is calculated, the FIR filter is ready for filter the noise.

4.2.2. Synchronization of signals

The signals from the rolling mill are not synchronized signals. The positions of sensors are shown in Fig. 5.1. It is clear that these signals do not start or end at the same time due to displacement between sensors. Thus a program is needed to synchronize all the signals. To complete the task, an algorithm is designed to remove non-signal data from each channel. The algorithm is done in two case.

(i). Forward pass.

In forward pass (pass 1, 3, 5, 7), we can easily decide where is the start of signal for several channels by using certain kinds of average criteria.

The first method is called five-point third-order smooth average.

Suppose there are n points sampled at equal intervals, $x_0 < x_1 <...< x_{n-1}$, the data are $y_0, y_1, ..., y_{n-1}$. We may use a polynomial equation to interpolate the data points.

$$y = a_0 + a_1 x + a_2 x^2 + a_3 x^3 \tag{4-3}$$

The coefficients may be calculated by the Least Square Method. The final equations are

$$\bar{y}_{i-2} = \frac{1}{70}(69 y_{i-2} + 4 y_{i-1} - 6 y_i + 4 y_{i+1} - y_{i+2}) \tag{4-4}$$

$$\bar{y}_{i-1} = \frac{1}{35}(2 y_{i-2} + 27 y_{i-1} + 12 y_i - 8 y_{i+1} + 2 y_{i+2}) \tag{4-5}$$

$$\bar{y}_i = \frac{1}{35}(-3 y_{i-2} + 12 y_{i-1} + 17 y_i 128 y_{i+1} - 3 y_{i+2}) \tag{4-6}$$

$$\bar{y}_{i+1} = \frac{1}{35}(2y_{i-2} - 8y_{i-1} + 12y_i + 27y_{i+1} + 2y_{i+2}) \tag{4-7}$$

$$\bar{y}_{i+2} = \frac{1}{70}(-y_{i-2} + 4y_{i-1} - 6y_i + 4y_{i+1} + 69y_{i+2}) \tag{4-8}$$

For the first two points and the end two points, eq.(4-4) (4-5)and (4-7) (4-8) are used for computing. In the program in Appendix D, the subroutine *smf( )* calculates the result. This method would minimize the influence by unexpected noise signals. Then it is easy to judge the start point and end point of a certain bar.

Another average method used is the simple 5-point average. This will also be used as a criterion.

The RE Gap, RE Force, RE Speed and RE Current are synchronized signals. Thus after identifying the start and finish of RE Force, other three signals can be obtained. Similarly, RR Force D/S side will determine RR Force O/S side, RR Top Current, RR Bottom Current and RR Speed. RR Exit temperature can also be obtained using the judging method. The Entry and Exit Width are computed based on a time lag using

$$Timelag = \frac{Distance}{\bar{V}_{rolling}} \tag{4-9}$$

where Distance is position difference between two signals, $\bar{V}_{rolling}$ is the average rolling speed/slab moving speed at that time. The difference between Entry Width and Exit Width is that the Entry Width uses a time lead while Exit Width has a time lag.

The procedures for forwarding pass are shown in Figure 4.6.

Forward pass

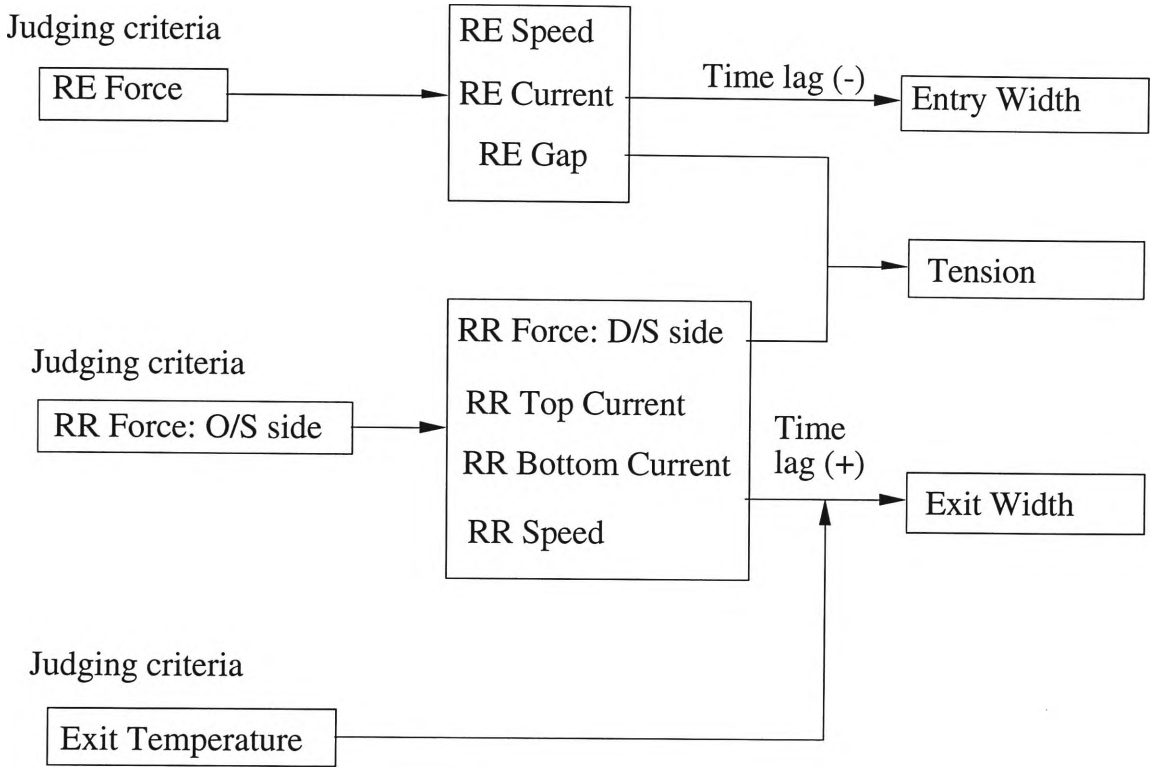


Fig. 4.6 Illustration of signal processing in forward pass

(ii). Reverse passes

The processing of reverse passes is similar to the forward passes, except that the direction is the other way and more signals are computed by means of time lag. Since it is difficult to judge the signals at RE in reverse pass, it is done by computing time lag from RR signals. (Most signal is not significant for RE in reverse passes).

After the signals are synchronized, it is necessary to figure out how much data is needed to be deleted from the head and tail of the file. It is because the data obtained are for the whole bar, while we only need the in-bar body part for width modelling. This is also computed in the time lag method mentioned above.

4.3 Data Preparation for Neural Network

Even after the data are synchronized, there remain things to be further processed, before these can be used to train the neural network. There may exist some observations that have abnormal values. Also some preprocessing of neural network data is needed.

Outliers are observations having a large influence on the least square estimator, which is known to be nonrobust. It is a focus of statistical diagnostics. There are many measures can be used as outliers detector. The field of diagnostics consists of a combination of numerical and graphical tools.

The measure adopted is the box plot, which is available in SAS. The upper and lower ends of the box indicate the 25th percentile and the 75th percentile. The line inside the box (with an asterisk at each end) indicates the median (the 50th percentile). The length of the box is one interquartile range, which is the difference between the 75th and the 25th percentiles. The + indicates the mean, which may be the same as the median but usually is not. The central vertical lines are called whiskers. Each whisker extends up to 1.5 interquartile ranges from the end of the box. Values outside the whiskers are marked with a zero or an asterisk. Values that are far away from the rest of the data are called *outliers*.

The general form of statements to produce Boxplot is shown in Panel 4.1

Panel 4.1 SAS program for box plot

```
infile 'c:\p3.nna';
array v{13} vr1-vr13;
input id v{*};
run;

proc univariate data=pass3 plot;
var vr1-vr13;
run;
```

The example of box plot is shown in Panel 4.2. Thus obtaining the value for quartiles.

Panel 4.2 Example of SAS box plot result

```
                          Univariate Procedure

Variable=VR1

                              Moments

N                 776    Sum Wgts          776
Mean         4.404318    Sum          3417.751
Std Dev      0.175772    Variance     0.030896
Skewness     -0.05415    Kurtosis     -0.11767
USS          15076.81    CSS          23.94419
CV           3.990897    Std Mean      0.00631
T:Mean=0     698.0079    Pr>|T|         0.0001
Num ^= 0          776    Num > 0           776
M(Sign)           388    Pr>=|M|        0.0001
Sgn Rank       150738    Pr>=|S|        0.0001

                          Quantiles(Def=5)

 100% Max       4.897       99%       4.836
  75% Q3        4.536       95%       4.658
  50% Med       4.408       90%       4.617
  25% Q1        4.275       10%       4.172
   0% Min       3.948        5%       4.133
                             1%       3.958
```

```
Range          0.949
Q3-Q1          0.261
Mode           4.333

                    Extremes

  Lowest    Obs      Highest    Obs
   3.948(    192)     4.89(     652)
    3.95(    194)     4.89(     654)
   3.955(    196)    4.893(     651)
   3.955(    195)    4.895(     655)
   3.955(      4)    4.897(     653)

                    The SAS System
                  Univariate Procedure
Variable=VR1

                 Histogram                    #          Boxplot
4.875+**                                      6             |
     .**                                      6             |
     .**                                      4             |
     .**                                      4             |
     .*************                           34            |
     .***************                         40            |
     .******************************          90            |
     .********************                    59          +-----+
     .*********************************       103         |     |
     .***************                         44          *--+--*
     .*************************               79          |     |
     .************************              72          |     |
     .******************************        93          +-----+
     .*****************                       46            |
     .*****************                       47            |
     .*********                               27            |
     .**                                       5            |
     .                                                      |
     .******                                  16            |
3.925+*                                       1             |
      ----+----+----+----+----+----+----+
      * may represent up to 3 counts
```

The detection of outliers using box plot theory is as follows.

Suppose

Q_1= 25th percentile

Q_2= 50th percentile

Q_3= 75th percentile

A point x is determined as an outlier if the following holds: .[2]

$$x < Q_1 - 1.5 * (Q_3 - Q_1) \tag{4-10}$$

$$\text{or } x > Q_3 + 1.5 * (Q_3 - Q_1) \tag{4-11}$$

Then the data point x is an outlier

After the diagnosis of outliers, the data is subjected to the format to be used in neural network. As the network would need data in the range of [0,1] or [-1, +1], we need to normalize all the data. The transformation of the data is by the following formula

$$x_{normalized} = \frac{H - L}{Max - Min} * x + \frac{L * Max - H * Min}{Max - Min} \tag{4-12}$$

where Max, Min are the range of data before normalization, H, L are the range of target, $x_{normalized}$ is the data point value after normalization.

This can also be done by using NeuralWare facility of MinMax table. Only the Min and Max value need to be supplied. Hereby, these values are set in accordance with the

[2] This outlier definition is in accordance with BHP SPPD HSM.

values to detect outliers. The upper boundary and lower boundary correspond to the range of outlier.

Sometimes, it could be necessary to reduce the amount of data to be used for network modelling. As the frequency of sampling is set to meet the requirement of sampling theorem, it is much higher than the signal frequency. This can give redundant information for the neural network, when they may have identical observations. A solution of this is to reduce the data frequency. This could make the network trained on valid, but less amount of data. The whole data processing procedure is shown in Fig. 4.7.

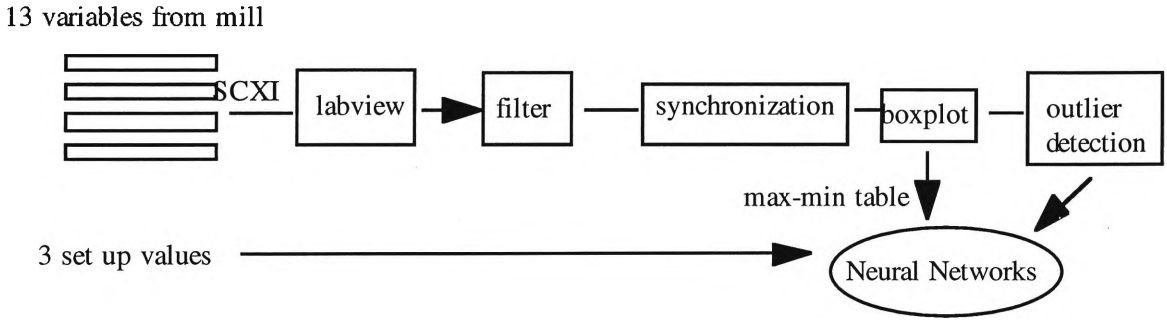


Fig. 4.7 Data logging and processing for neural network modelling

Chapter 5

Neural Network Width Model & Prediction

A model is, of certain accuracy and range, which mimics certain aspects of observed behavior, thus enabling useful prediction to be made, either qualitative or quantitative. In this chapter, the need and objective of neural network model for in-bar width prediction will be discussed. The structure of such a neural network model is given. Finally, some primary results about width prediction are presented.

5.1 Aim of neural network model for in-bar width application

The aim of modelling is to increase the operating range and accuracy of a particular process, rolling process. Modelling also wants to solve why unexpected difficulties have arisen. Recalling the problems existing in Roughing Mill width control, there is no physical model that suits accurate in-bar width control, although various empirical equations exist long ago. Empirical equations may tell the trend of width change as an

estimate, but it is so inaccurate that it may differ from one mill to another due to different mill arrangement.

From all the efforts to model the mill, we know that it is not easy to set an accurate physical model for the rolling mill, especially its width change model. The reason is that the rolling process is a complex physical process with many sub-processes not fully understood.

The use of Neural Networks in mathematical modelling provides an alternative width deviation model for the roughing mill. With neural networks, hysical mechanisms for complex processes are not required to be fully understood. Traditionally, it would be impossible to introduce any physical model without expert knowledge about the process being modelled. All the computation is taken with strictly defined mathematical terms. All the complexity may be overcome with the use of Artificial Intelligence, especially neural networks for modelling.

5.2 The design of neural network application

With neural network, no specific knowledge about the process in rolling deformation is required. The basic thing to know is that the slab with an entry width will come out with an exit width after going through the Roughing Mill. This is like a set of paired numbers. The neural network is supposed to learn all the facts about the paired numbers. Then for a new input number, it will predict its corresponding output number. The exit width is not just determined by the entry width. as there are factors related to the deformation of slab. From chapter one, it was discussed that other factors like rolling temperature, rolling gap setting, speed, etc., also effect in the process. Thus, in the design of neural network

model for the width deviation, all these factors have to be taken into consideration. The goal of the neural network is to learn all the relationship of different variables with regard to the exit width change.

After a study on all the possible variables, 12 input variables were chosen to model the exit width deviation. The 12 variables and the exit width deviation are sensor signals. They are transmitted from the sensors or gauges mounted around the roughing mill. These signals are transferred to the existing control PLC arrays, where all the signals can be obtained through the test points on the control panel. The detail of the data logging and preliminary signal processing was discussed in the chapter 4.

The 12 variables are as listed in table 5.1. The positions of the variables on the roughing mill are illustrated in Fig. 5.1 .

Table 5.1 Select signals for neural network input & output

Variable No.	Signals
1	Entry Width
2	RE Gap Deviation
3	RE Force
4	RE Speed
5	RE Current
6	RR Bottom Speed
7	RR Exit Temperature
8	RE-RR Tension
9	RR D/S Force (Drive side)
10	RR O/S Force (Operator side)
11	RR Top Current
12	RR Bottom Current
13* output	Exit Width

The entry width is an inevitable variable. The edger gap is another important one, since the gap deviation will delimit any change of slab width. All other ones are relating to the rolling process. The current reflect the output of motor power, and rolling forces are related to the plastic deformation and friction of rolling in some way. The speed also plays a role in width change. The variables from the reversing mill are included for the reason that there is also a width change at the Roughing Mill Rougher when it is carrying out thickness control. Thus, the rougher also contributes to the exit width change. The temperature of the slab is related to deformation, thus the width deformation. The tension contributes to the plastic deformation of the material. All these variable are considered to have varying influence on the width change.

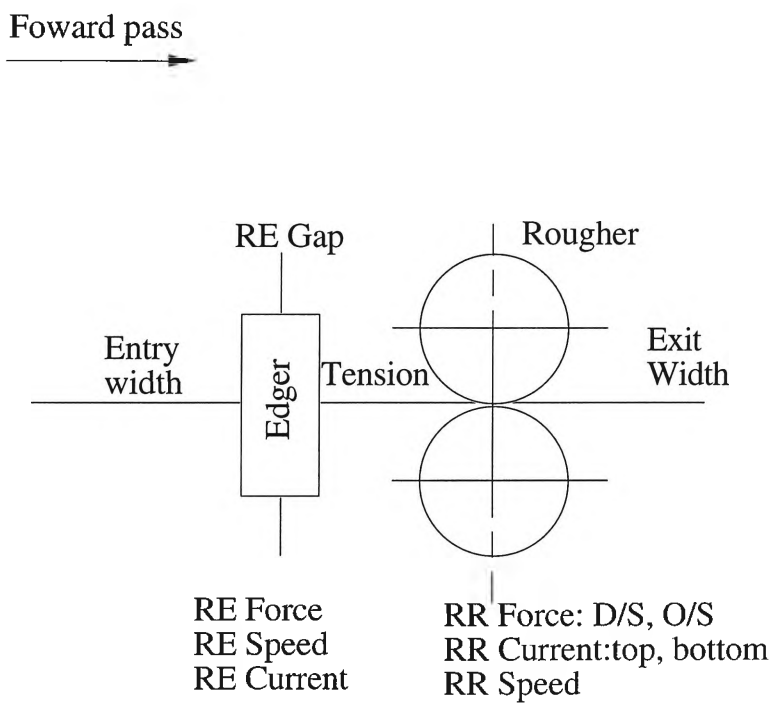


Fig. 5.1 Neural Network input signals from the roughing mill

It is not required to quantitatively know how the variables affect the change, and in what way, since the Neural Network can learn the facts regarding the variation of the 12 input signals and the output signal (exiting width).

Neural Network modelling of the Roughing Mill width deviation is then carried out. The Back-Propagation neural network was selected as the tool to fulfil the modelling task as it is the most widely used neural network and suitable for modelling and pattern recognition. The supervised learning structure is shown in Fig. 5.2. The structure of such a neural network would be like the general form shown in Fig. 3.8, in which all the variables correspond to the nodes. There is only one hidden layer in the neural network was designed as it was considered to be sufficient to provide the nonlinearity required. In Fig. 5.3, not all the hidden nodes are drawn due to its relatively large number. At present, the number of nodes of the hidden layer is not a fixed number for different cases in width modelling for the variation of training size. This is discussed this later in this section.

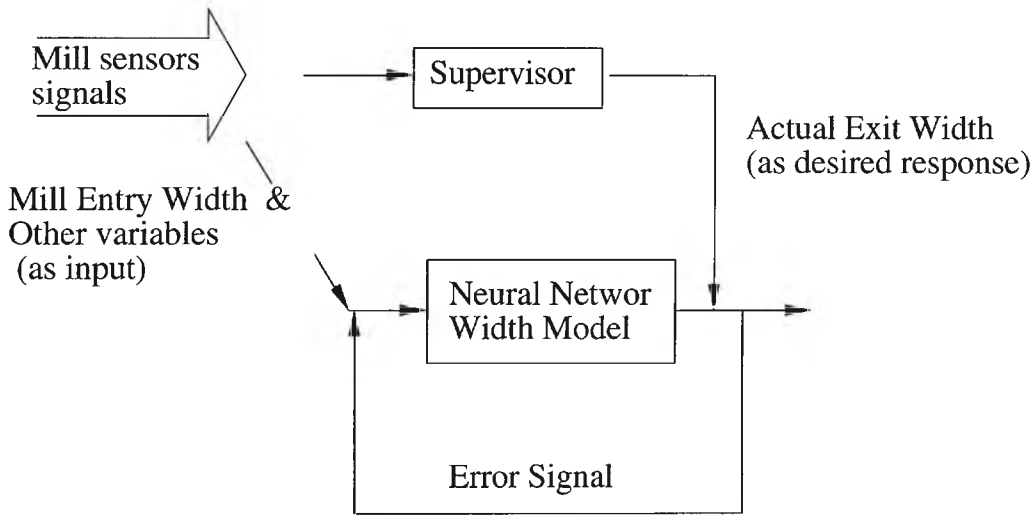


Fig. 5.2 Neural network supervised learning of width deformation

There are seven passes for the slab to pass the Roughing Mill. The 1st, 3rd, 5th and 7th passes are the forwarding passes, where as the 2nd, 4th, 6th passes are the reverse passes. The neural network is designed to learn specificly about each single pass, and therefore is pass-based. The reversing passes are not used for a width control as the edger rolls are not used to carry out width reduction. Only in the 1st, 3rd, 5th and 7th passes, there is a need to model the width change. For a specific section of the same grade of slabs, they have similar entry width and exiting width. Given their signals during rolling

to train the neural network, the neural network can learn from all the data and find a relationship of inputting signals and output signal for each pass.

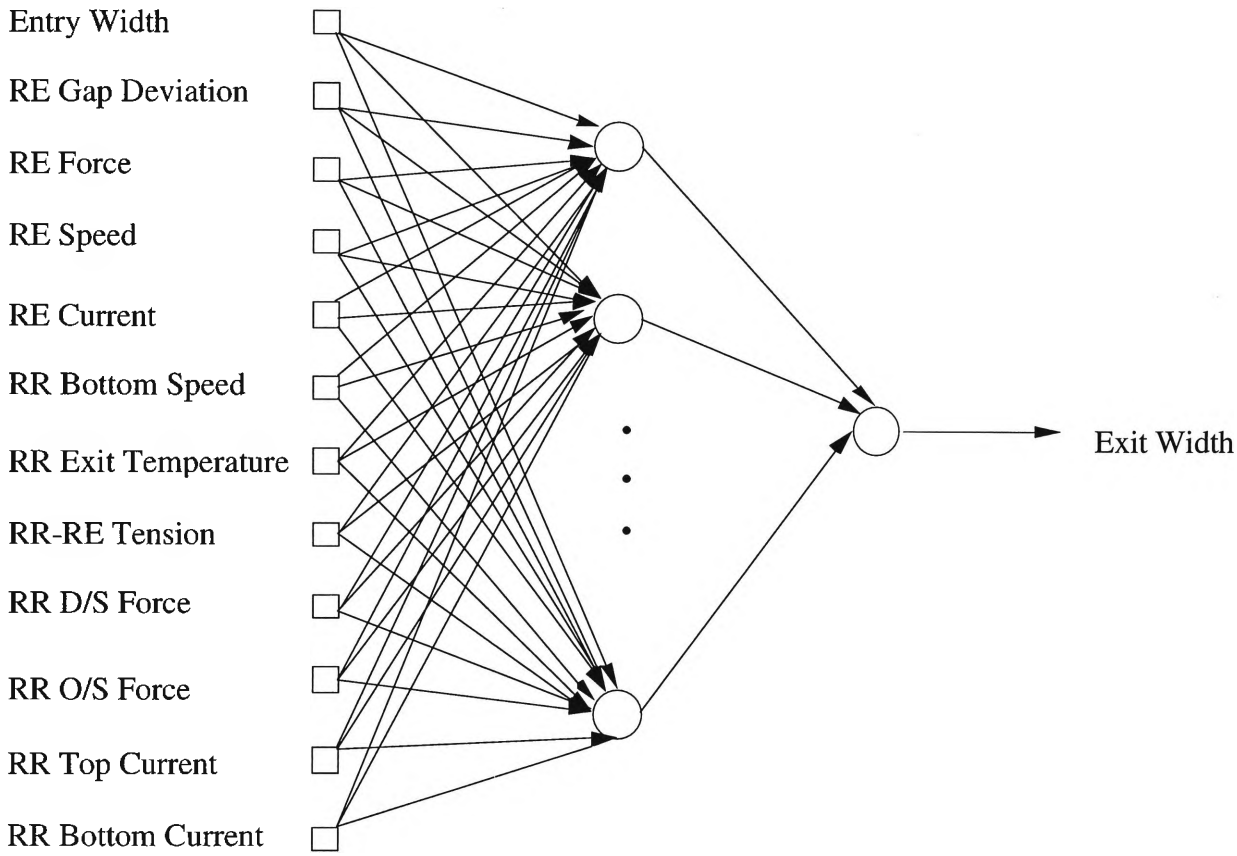


Fig. 5.3 Back-propagation neural network for width deviation model

5.3 The neural network program

To compute the neural network training and testing, we use the software, NeuralWorks Professional II/Plus, from NeuralWare, Inc. This saved a lot of time as it is a versatile software. With this software, many kinds of neural networks are available. It also allows the user to modify the network parameters. In Appendix A, a brief introduction to the software will be given.

To acquire an accurate model, the neural network has to be trained with all the facts (data) concerning width change from the roughing mill. The number of neurons in hidden layer is determined by the sample size of training data. It is suggested that the approximate number of hidden neurons [1] should be

$$N = \frac{number \quad of \quad training \quad cases}{5*(m+n)} \tag{5-1}$$

where:

- cases is the number of records in the training file.
- m is the number of neurons in the output layer.
- n is the number of nodes in the input layer.
- N is the number of neurons in the hidden layer.

Once the training data is set, the network can also be built up as shown in Fig. 5.4. At the start of training process, the network is initialized. As discussed in section 3.4, the initialization play an important role in network convergence. The software provide a choice of Gaussian or uniform distribution. From the results, the network with a Gaussian initialization learns better.

[1] It is suggested in the manual, Using NeuralNetworks, NeuralWare, UN-20.

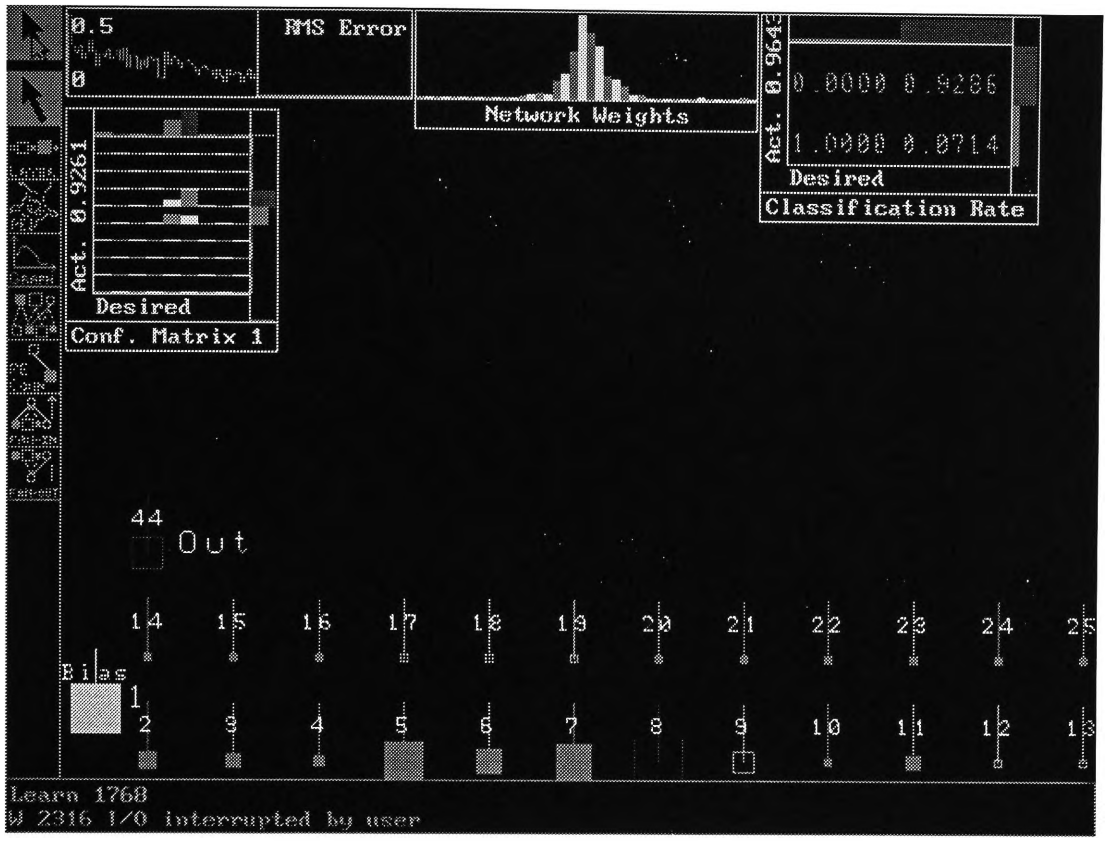
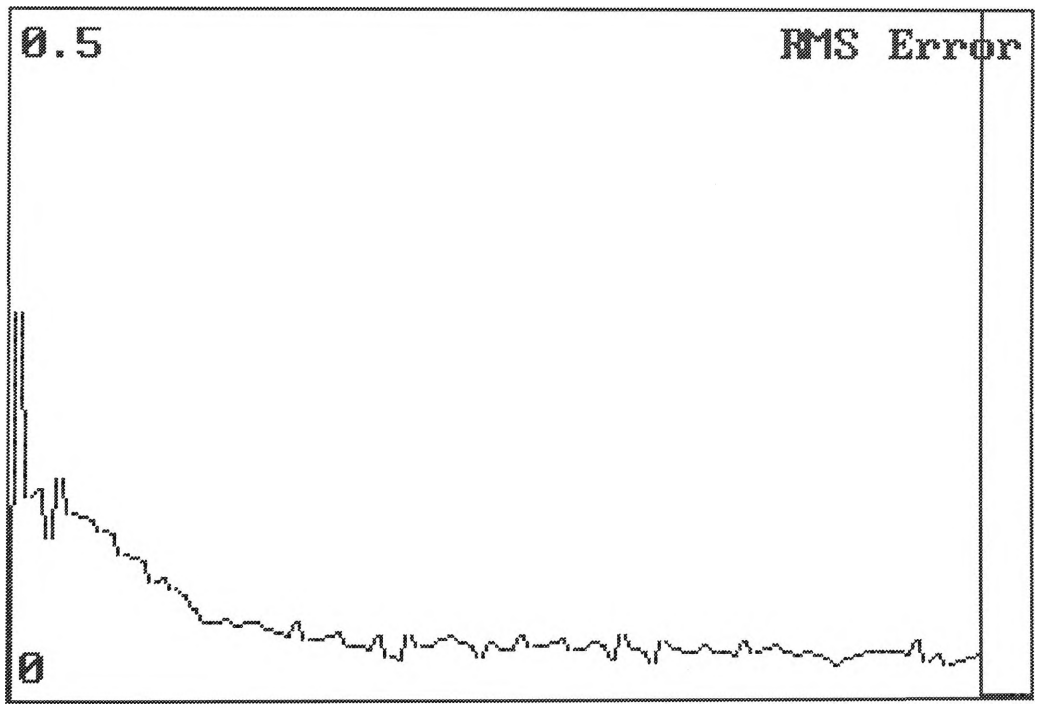


Fig. 5.4 The neural network model in training



time

Fig. 5.5 The RMS error change when start Neural Training

The error of the network is computed in terms of rooted mean square (RMS) error, which is the square root of the cost function used in chapter 3 for error-correction. The square root removes the magnitude increase of the squaring operation. This RMS error is a valuable and common measure of the performance of a network during training. At the beginning of training, the error drops sharply as shown in Fig. 5.5. As the training goes, it stabilise to some constant. To reduce the error, it is vital to select the right transformation function, learning schedule and epoch size.

(i) The transformation function.

The selection of sigmoid transformation functions in 3.1.2 is based solely on whichever has a better performance for the network. The logistic function is set as the default function. However, when the network learns slowly, the hyperbolic tangent function can be the alternative for trial.

(ii) The learning schedule.

The learning rate and momentum are primary parameters in a learning schedule for the neural network (Back-propagation here). From the neural network theory, the learning rate is set to a constant for the whole network throughout the time of training. In section 3.4, it has been stated that the learning rate for the output layer is lower than that of the hidden layer. Other than this, it is a good idea to assign different learning rate along the time axis (i.e., iteration of training), for a layer of neurons. This is because the network goes toward the minimal along the error surface. When it is near the minimum, the network may miss the point, if the step of the network modification is too large. On this occasion, the learning rate should be reduced, thus the step would be shorter and possibly find the minimum. Also, a low learning rate can make the learning trajectory very smooth. However, it is not desired to have a low learning rate at the beginning, which would make the learning too slow.

Similar to the learning rate, the momentum is also subjected to changes in the learning schedule. In Fig. 5.6, a screen from the training of width model shows the editing of the learning schedule.

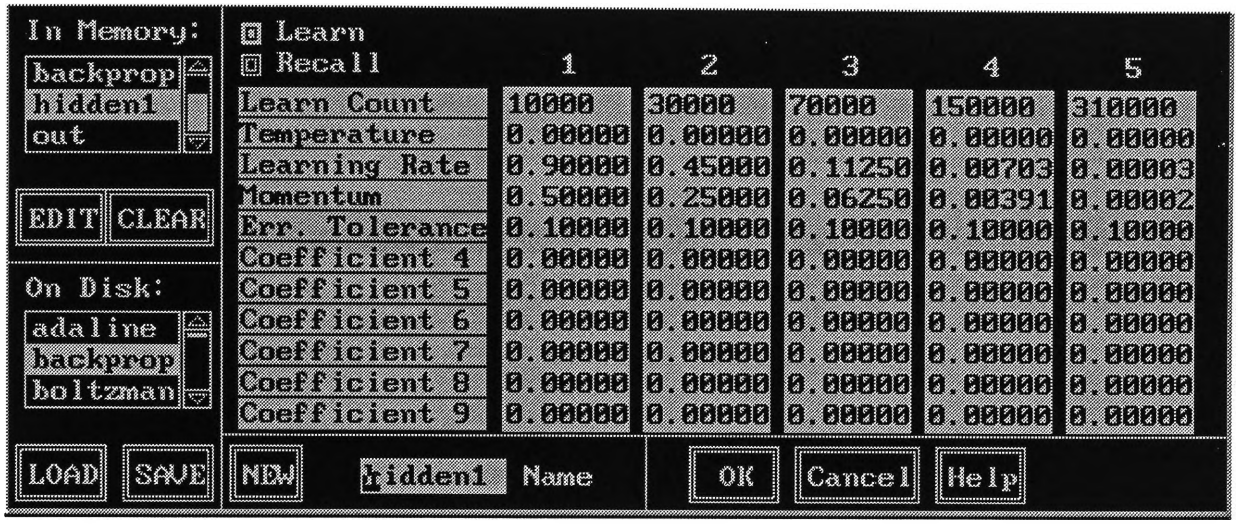


Fig. 5.6 Editing screen of learning schedule

(iii) Epoch size.

An epoch is the number of training presentations between weights updates. In a batch mode training, the errors are accumulated over an epoch before the synaptic weights of the network make adjustment. A epoch size greater than 1 means that, the network learn over a sample size each time, which has an average effect. However, the epoch size should be chosen properly for a large epoch size can cause slow learning or slow convergence of the network.

Training of a neural network is more like a trial-and-error method than a standard procedure. The network parameters provide lots of options. Sometimes, it takes a long time to find out which parameter need to be modified.

5.4 Testing of the neural network model for width deviation

To determine how well the model accuracy is and whether it needs improvement, a test phase was implemented. The test phase is one way of determining how well the network has learnt, and how well the network will perform. Since training time is rather application specific. The training time is based on the performance required for the network.

The common practice is to set aside a percentage of examples to serve as test cases. The network is trained on the training cases, and then the test cases serve as a way of measuring network performance. Note that the test case and training cases are two samples from the same source. During the test phase, the input data are presented to the network and the network provides predicted output results. As the signals were obtained from the Roughing Mill, the actual exit width deviation is already known, which is the measured response for the network. To indicate how well the network will perform its desired task, it can be done by comparing the predicted response from the neural network model and the measured response.

A major difference between training and testing is that in the test case, the weights in the network are not updated. The weights will be set at the values right after the training of the network. In Fig. 5.7, the test phase is described.

On testing of neural network model for in-bar width, a new slab other than those for training is used. The following example is the network trained for a section of slabs, of a plain carbon manganese grade, aimed dimensions are 908 mm x 25.0 mm x 122 m (width, thickness, length). The coil number of those used for training are I6165, I6167,

I6171, I6172, I6173, I6174, I6175, I6176, I6178. After training, we use slab I6179 to test the result of the training.[2]

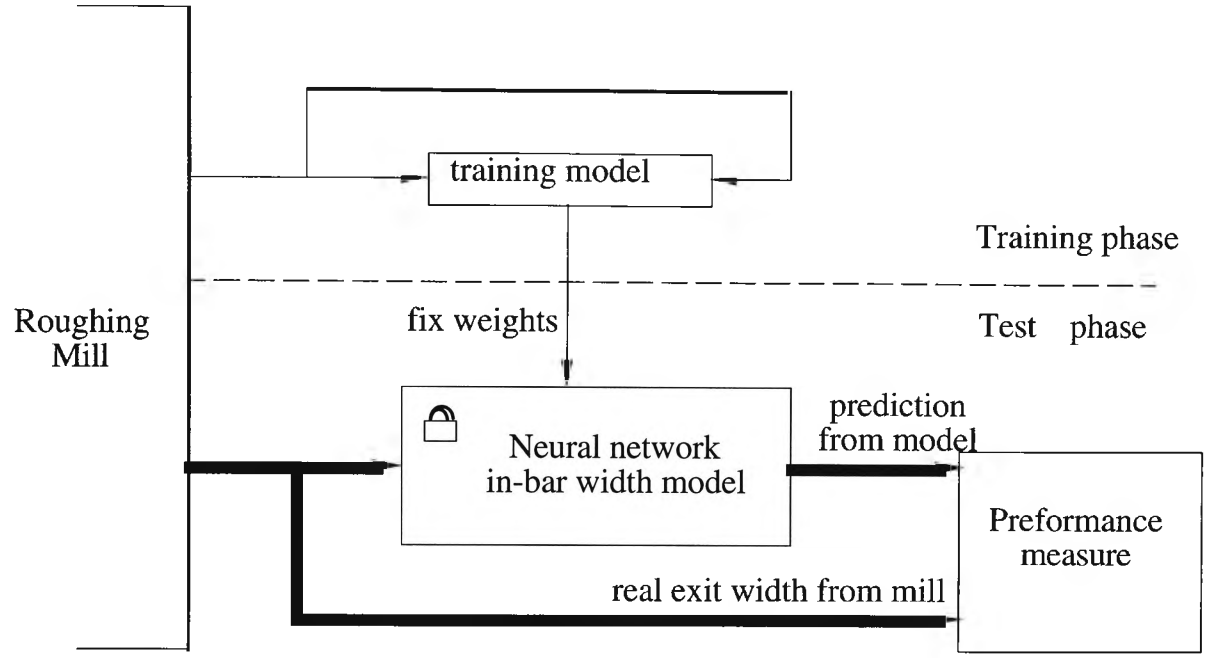


Fig. 5.7 The test phase of neural network model

Table 5.2 Sample test result for mill width model

Measured Width	Predicted Width (NN)	Measured Width	Predicted Width (NN)
1.165	1.423	2.085	1.777
1.275	1.359	2.010	1.806
1.265	1.386	1.960	1.882
1.300	1.403	2.010	1.871
1.315	1.432	1.985	1.840
1.605	1.408	2.010	1.856
1.545	1.322	1.995	1.919
1.645	1.227	1.705	1.878
1.580	1.338	1.665	1.838
1.825	1.437	1.705	1.892
1.730	1.690	1.665	1.841
1.825	1.726	1.705	1.618
1.850	1.634	1.665	1.651
1.910	1.551	1.705	1.635
1.875	1.572	1.705	1.858
2.120	1.643	1.595	1.790
2.060	1.738	1.595	1.847
2.105	1.749	1.620	1.769

[2] Due to noise in pass 1, we only designed network for pass 3, pass 5 and pass 7.

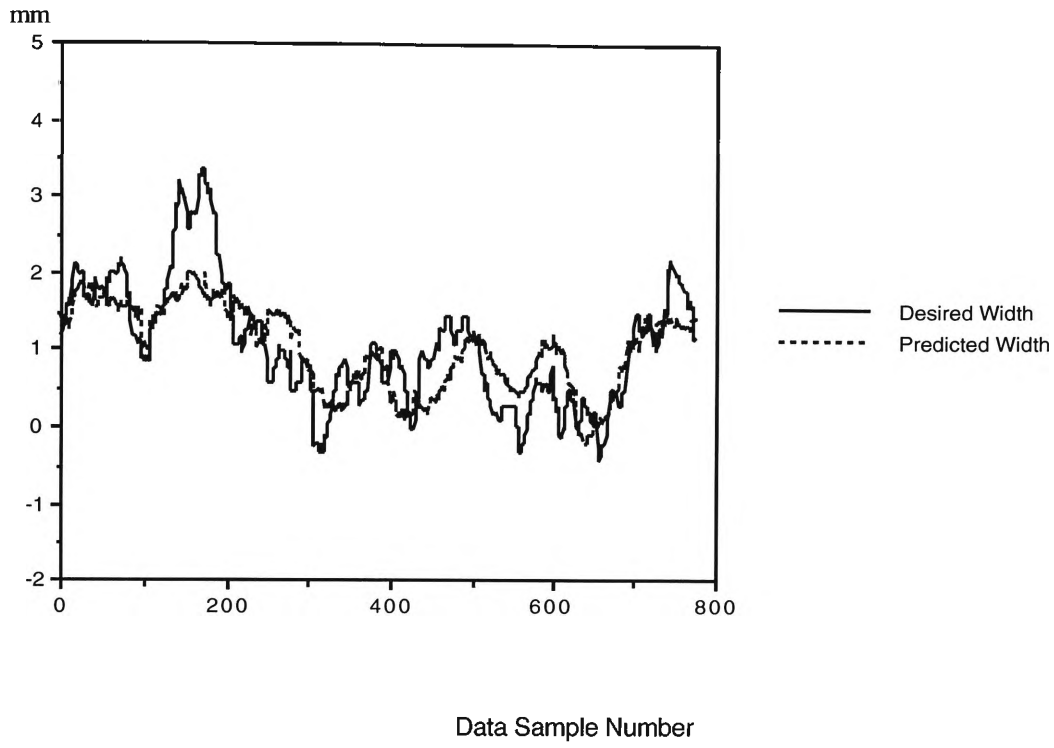


Fig5.8 Test result of I6179, pass 3
(Desired width is the measured width)

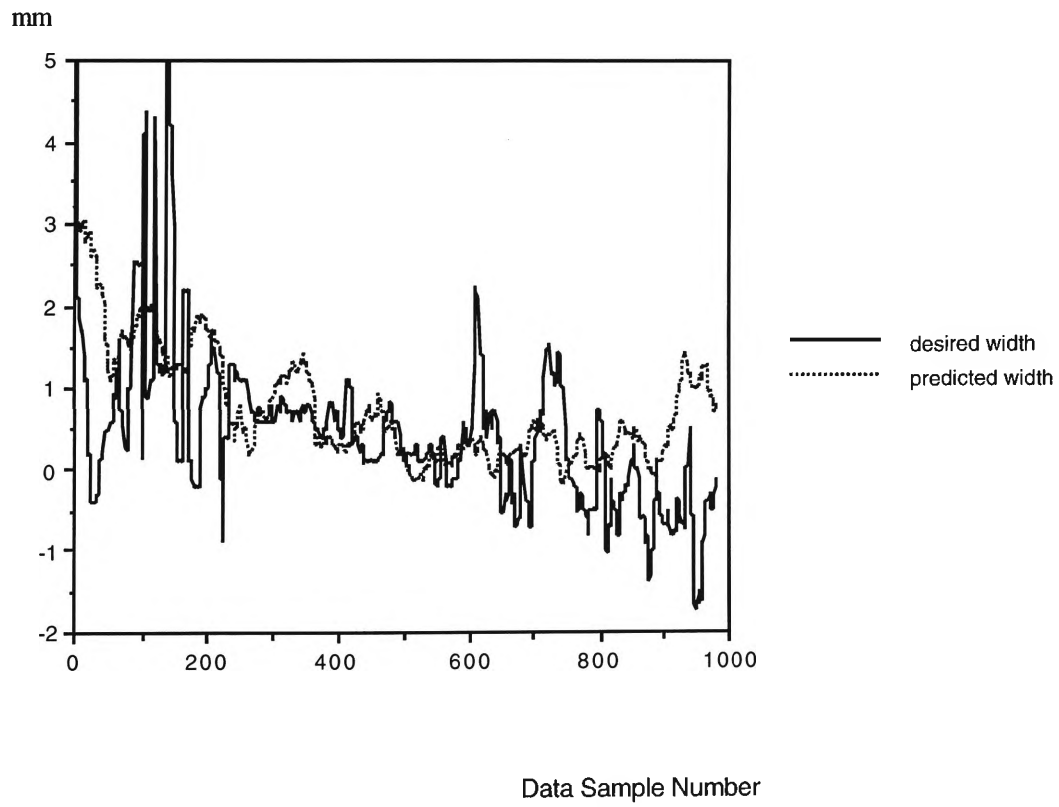


Fig. 5.9 Test result of I6179, pass 5
(Desired width is the measured width)

Table 5.2 is a layout of the test result for pass 3 of the above network. Due to limit of length, it is just a part of the test file. Fig. 5.8 shows the test result of pass 3. Fig 5.9 shows the test result of pass 5.

The figures show a fair interpolation of the desired response (measured width). The result indicates that it is well trained neural network. However, it is not convincing enough just by the graphic result to prove the neural network model's accuracy. A quantitative criteria is needed for the examination of the model. This is discussed in next section.

5.5. Statistical Test on model examination

The practical verification method is to use statistical t-test and F-test to examine the model mean and variance[3]. These are standard statistical tests being used here to give a quantitative proof for the neural network models.

(i) Test of Significance for means:

The hypothesis for test means are,

$$H0: \mu_1 = \mu_2$$

$$H1: \mu_1 \neq \mu_2$$

H0 is the Null Hypothesis. H1 is the Alternative hypothesis. With level of Significance α, the Criteria Zone is that , Reject H0 if $Z > Z_0$ or $Z < -Z_0$

$$Z = \frac{\bar{x} - \mu}{\sigma / \sqrt{n}} ..$$

where $\bar{x}$ is the sample average for μ, σ is the variance, n is the sample size.

This is called t-test, because the Criteria Zones are based on t-distribution.

(ii) Test of Standard Deviation Significance:

---

[3] Communication with D Frances from BHP Steel

The Null hypothesis is H0: $\sigma_1 = \sigma_2$

σ_1= standard deviation of the difference between the aimed and observed value;

σ_2= standard deviation of the difference between the predicted and observed value.

Alternative Hypothesis, H1: $\sigma_1 \neq \sigma_2$.

Level of significance: α

Criterion: Reject the null hypothesis H0, if $F > F_0$ and $F < 1/F_0$

$$F = \frac{(S_1)^2}{(S_2)^2}$$

where S are variances. The criteria value for F are indexed in the F-distribution table.

The result from a good model should satisfy the following requirement:

(I) There is no significant difference between Measured and Predicted mean width value.

(ii) There should be a significant difference between the two deviations, in that the predicted width is more consistent in predicting the observed width than the aimed width.

The result of above model tested on I6179, pass 3 is:

Mean (Aimed-Observed) =1.062 , Standard Deviation = 0.805

Mean (Predicted - Observed)= 0.0447, Standard Deviation= 0.514

When α=0.01, Criteria zone for mean is 2.576. The Z value for Aimed-Observed is 36.7, which means rejection. There is a strong suggestion that the Observed value is different from Aimed value. The Z value for (Predicted - Observed) is 2.425, which is acceptable. This means, the model successfully follows the mean.

Its F test value is 2.45, the criteria zone is between 0.82 and 1.22. This test reject the null hypothesis, and the standard deviation between (aimed - observed) is different to that between (predicted - observed).

But some time, the results may vary. It may not pass the t-test and F-test. What can be done to improve it, is the topic we are going to discuss in the following chapters.

Chapter 6

STATISTICAL ANALYSIS for NEURAL NETWORK WIDTH MODEL

Neural network is a good tool for modelling complex processes. Yet sometimes there may be some significant errors due to neural network's inability to model all the process features. This Chapter deals with the statistical analysis applied on the neural network width parameters, and makes some suggestions to improve the network.

6.1 Aim of statistical analysis

Although most of the time neural networks give good interpolated predictions on width deviation, there exist some biased predictions for width. The source of errors may be caused by several reasons. One possible reason is redundant variables and thus colinearity among variables. The colinearity could in turn give neural network false

information about the relationships among variables. The result is then a biased model after training.

As we know, the Neural Network is in nature a different path of statistics (White, 1990 and Geman et al., 1992). It then comes the idea of using statistical analysis to analyse all the width related rolling variables which are used in neural network model.

The data analysis will optimise Neural Network structure and make input variables independent. The analysis will also reduce the dimensions of the problem without loss of useful information. From the analysis, we can understand the degree of importance of different processvariables to the width variation at the exit. Some suggestion can be made on how to improve the network.

6.2 Regression analysis theory

Regression analysis is the statistical methodology for modelling and predicting values of one or more variables, known as *dependent* variables, from predictors (*independent*) variables. It can also assess the predictor variables effects on the response (*dependent*) variables.

6.2.1 Classical Linear Regression Model

Assume v_1, v_2, ..., v_r to be r predictor variables that are considered to be related to a response variable Y. The classical linear regression model states Y is dependent on the continuous manner of the v_i and random error ε. As the values of predictor variables are fixed after obtained from an experiment, the error is viewed as a random variable with a certain distribution.

The model of regression is given below:

$$Y = \beta_0 + \beta_1 v_1 + \ldots + \beta_r v_r + \varepsilon \quad (6\text{-}1)$$

$$\text{Response Var} = \text{Mean (dependent on } v_1, v_2, \ldots, v_r) + \text{error} \quad (6\text{-}2)$$

This model is a linear function of all unknown parameters β_0, β_1,..., β_r. But predictor variables v_i may or may not be of the first order in the model.

After an experiment, with n independent observations on Y and the associated v_i obtained, the complete model yields n equations.

$$\begin{aligned} Y_1 &= \beta_0 + \beta_1 v_{11} + \beta_2 v_{12} + \cdots + \beta_r v_{1r} + \varepsilon_1 \\ Y_2 &= \beta_0 + \beta_1 v_{21} + \beta_2 v_{22} + \cdots + \beta_r v_{2r} + \varepsilon_2 \\ &\vdots \qquad\qquad \vdots \\ Y_n &= \beta_0 + \beta_1 v_{n11} + \beta_2 v_{n2} + \cdots + \beta_r v_{nr} + \varepsilon_n \end{aligned} \quad (6\text{-}3)$$

The errors are assumed to have the following properties:

1. $E(\varepsilon_j) = 0$;

2. $Var(\varepsilon_j) = \sigma^2$;

3. $Cov(\varepsilon_j, \varepsilon_k) = 0$, j≠k.

In matrix notation, the regression model is expressed as:

$$\mathbf{Y} = \mathbf{V}\ \beta + \varepsilon \quad (6\text{-}4)$$

and $E(\varepsilon) = 0$, $Cov(\varepsilon) = \sigma^2 I$, where β and σ^2 are unknown parameters and the design matrix **V** has jth row $[v_{j0}, v_{j1}, \ldots, v_{jr}]$.

6.2.2 Least-Square Estimation

Regression analysis is used to solve the equations of the classical model which attempts to find a suitable curve(model) to fit the original data. The least squares estimation is a method employed to select **b** (as an estimation of β), which minimizes the sum of squared differences

$$S(\mathbf{b}) = \sum_{j=1}^{n}(y_j - b_0 - b_1 v_{j1} - \ldots - b_r v_{jr})^2$$

$$= (\mathbf{y}-\mathbf{Vb})'(\mathbf{y}-\mathbf{Vb}) \qquad (6\text{-}5)$$

The estimated parameters **b** is chosen by the criterion of Least Square Estimation. It is considered as unbiased estimation of β, denoted as $\hat{\beta}$.

Let **V** have full rank r+1≤n. The least squares estimate of β in the classical regression model is given by

$$\hat{\beta} = (\mathbf{Z}'\mathbf{Z})^{-1}\mathbf{Z}'\mathbf{y} \qquad (6\text{-}6)$$

The residuals is calculated as

$$\hat{\varepsilon} = \mathbf{y} - \hat{\mathbf{y}} = [\mathbf{I} - \mathbf{V}(\mathbf{V}'\mathbf{V})^{-1}\mathbf{V}']\mathbf{y} \qquad (6\text{-}7)$$

Also, the

Residual Sum of Squares $= \sum_{j=1}^{n}(y_j - \hat{\beta}_0 - \hat{\beta}_1 v_{j1} - \cdots - \hat{\beta}_r v_{jr})^2 = \hat{\varepsilon}'\hat{\varepsilon}$

$$= \mathbf{y}'[\mathbf{I} - \mathbf{V}(\mathbf{V}'\mathbf{V})\mathbf{V}']\mathbf{y} = \mathbf{y}'\mathbf{y} - \mathbf{y}'\mathbf{V}\hat{\beta} \tag{6-8}$$

After decomposition, we obtain

$$\sum_{j=1}^{n}(y_j - \bar{y})^2 = \sum_{j=1}^{n}(\hat{y}_j - \bar{y})^2 + \sum_{j=1}^{n}\hat{\varepsilon}_j^2 \tag{6-9}$$

$$\begin{pmatrix} total\ \ sum \\ of\ \ squares \\ about\ \ mean \end{pmatrix} = \begin{pmatrix} regression \\ sum\ \ of \\ squares \end{pmatrix} + \begin{pmatrix} residual\ \ (error) \\ sum\ \ of\ \ squares \end{pmatrix} \tag{6-10}$$

By decomposition, the quality of the model fit can be measured by the coefficient of determination

$$R^2 = 1 - \frac{\sum_{j=1}^{n}\hat{\varepsilon}_j^2}{\sum_{j=1}^{n}(y_j - \bar{y})^2} = \frac{\sum_{j=1}^{n}(\hat{y}_j - \bar{y})^2}{\sum_{j=1}^{n}(y_j - \bar{y})^2} \tag{6-11}$$

The quantity of R^2 gives the proportion of the total variation in the y_j's attributable to the predictor variables.

Let $\mathbf{Y}=\mathbf{V}\beta+\varepsilon$, where $\mathbf{V}$ has full rank r+1 and ε is distributed as $N_m(0,\sigma^2\mathbf{I})$. The maximum likelihood estimator of β is the same as the least squares estimator, $\hat{\beta}$.

A $100(1-\alpha)\%$ confidence region for β is given by

$$(\beta - \hat{\beta})'\mathbf{V}'\mathbf{V}(\beta - \hat{\beta}) \le (r+1)s^2F_{r+1,n-r-1}(\alpha) \tag{6-12}$$

where $F_{r+1,n-r-1}(\alpha)$ is the upper (100α) percentile of an F-distribution with r+1 and n-r-1 degree of freedom.

6.2.3 Prediction and Model Modification

Once the regression model is found fitting satisfactorily with the data, it is then cast into two major domain of application. First, the model can be used to estimate the regression

function at certain point. Second, the model may predict or forecast a new observation at a point, which may be an existing point or a new one.

Let $\mathbf{v}_0 = [1, v_{01}, ..., v_{0r}]'$ be a set of selected values for predictor variables, $\hat{\beta}$ be the least square estimation and Y_0 denote the response when predictor variables have the value of $\mathbf{v}_0$. The expected value of Y_0 is

$$E(Y_0 \mid \mathbf{v}_0) = \beta_0 + \beta_1 v_{01} + ... + \beta_r v_{0r} = \mathbf{v}_0' \beta \tag{6-13}$$

Its least square estimation is $\mathbf{v}_0' \hat{\beta}$. When Y_0 is a new observation at $\mathbf{v}_0$, the prediction of Y_0 is more than a simple expected value. According to the regression model,

$$Y_0 = \mathbf{v}_0' \beta + \varepsilon_0 \tag{6-14}$$

where ε_0 is normally distributed as N(0, σ^2) and is independent of ε, $\hat{\beta}$ and s^2.

$$\mathrm{Var}(Y_0 - \mathbf{v}_0' \hat{\beta}) = \sigma^2 (1 + \mathbf{v}_0' (\mathbf{V}'\mathbf{V})^{-1} \mathbf{v}_0) \tag{6-15}$$

The above equation shows the variance of the forecast error, the scale of which reflects the model fitness and accuracy.

The difficulty of practical regression analysis lies in how to find out variables that are crucial for changes in dependent variable, delete insignificant ones and increase some variables of high order or special function, such as exponential or log function.

In fact, it is still a trial and error based technique, but there are some guidelines that can be used. One of the possible ways is to assume the present model "correct" and plot the residuals plots against predictor variables and response variables. From the shape of

various plot, it could then be decided if any items (e.g. high order terms) need to be added in.

Another way of doing the selection is called *step-wise* selection. This method applies well for selecting predictor variables from a large set. It is often difficult to formulate an appropriate regression function when a large number of variables are presented to the model all at once. A selection method is to try all the subsets of variables for regression, and check some criterion quantity such as R^2. Stepwise regression attempts to select important predictors without considering all the possibility. The method is to compare each variable's contribution to the response. If they are below the criterion value, it is then selected into the model.

A high R^2 does not necessarily mean a good regression model. There may be some potential problems for the model or even a misled biased model. One factor of great concern is the colinearity among variables. When $\mathbf{V}$ is not of full rank, some linear combination, such as $\mathbf{Va}$, must equal $\mathbf{0}$. In such a situation, it is said to be *colinear*. This in turn implies that $\mathbf{V'V}$ does not have an inverse. Although for our regression situation, it is unlikely that $\mathbf{Va=0}$. Yet it may be very close to $\mathbf{0}$ for linear combination of $\mathbf{V}$, the inverse matrix $(\mathbf{V'V})^{-1}$ is still numerically unstable. The direct yield from this is large estimated variance for the $\hat{\beta}_i$'s and it is then difficult to detect the significant β_i. A suggestion to the problem is to delete one of strongly correlated variables and use Principle Components of predictor variables, which are independent from each other. The Principle Components method on width model will be discussed in section 6.4.

6.3 Regression analysis for width model

As stated in the aim part of analysis, the reason for conducting statistical analysis is to simplify Neural Networks structure and improve accuracy.

6.3.1 Preliminary knowledge about data from mill

Signals from the roughing mill are collected through PLC's. The real time data is sampled in 100 Hz. After preprocessing, files that contains observations of all width related variables are obtained. The variables, as listed in Table 5.1, are all related to exit width of the rolled slabs. Here it is relisted as Table 6.1 and assign variable number for each signal for later use.

Table 6.1 Signals used in statistical analysis

Variable No.	On Board Channel	Panel	Test Point	Variables
1	0	RMW	0	WG1 Deviation (Entry)
13	1	RMW	1	WG2 Deviation (Exit)
2	2	RMW	2	RE Gap Deviation
3	3	RMW	3	RE Force
4	4	RMC	0	RE-Speed
5	5	RMC	1	RE Current
6	6	RMC	2	RR Bottom Speed
7	7	RMC	3	RR Exit Temperature
8	8	RMC	4	RR-RE Tension
9	9	RMC	5	RR D/S Force
10	10	RMC	6	RR O/S Force
11	11	RMC	7	RR top Current
12	12	RMC	8	RR Bottom Current

Some pairs of variables are correlated or very close to each other. For example, the RE-Speed and RR Bottom Speed looks similar in shape as shown in Fig. 6.1. The difference between these two speeds in the forward passes determines the tension between the Roughing Edger and Reversing Rougher. The two force variables of RR on both sides of rolls, as illustrated in Fig. 6.2, although similar, show some difference.

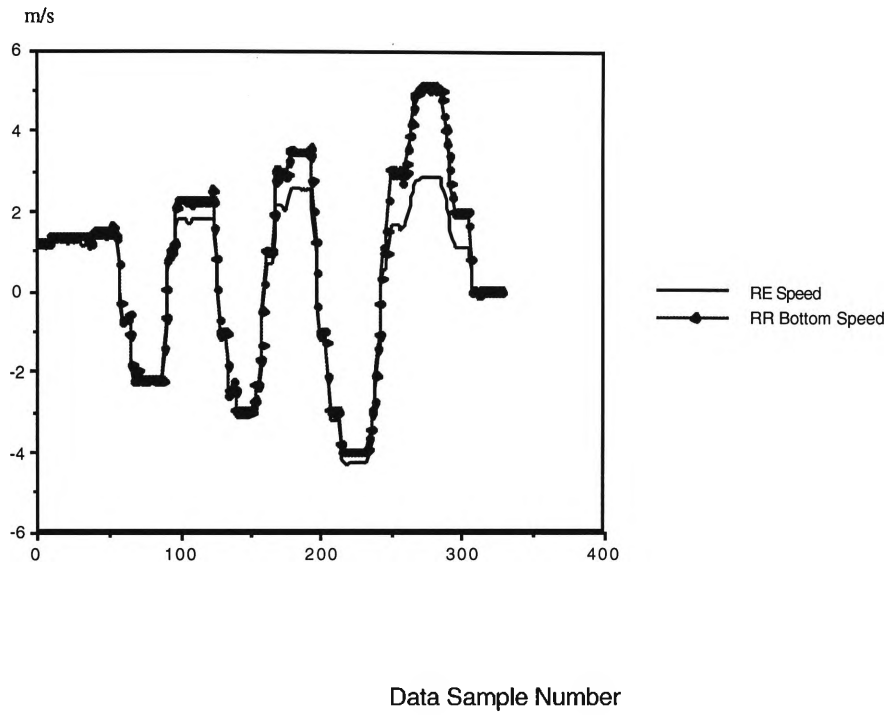


Fig. 6.1 RE speed and RR bottom speed

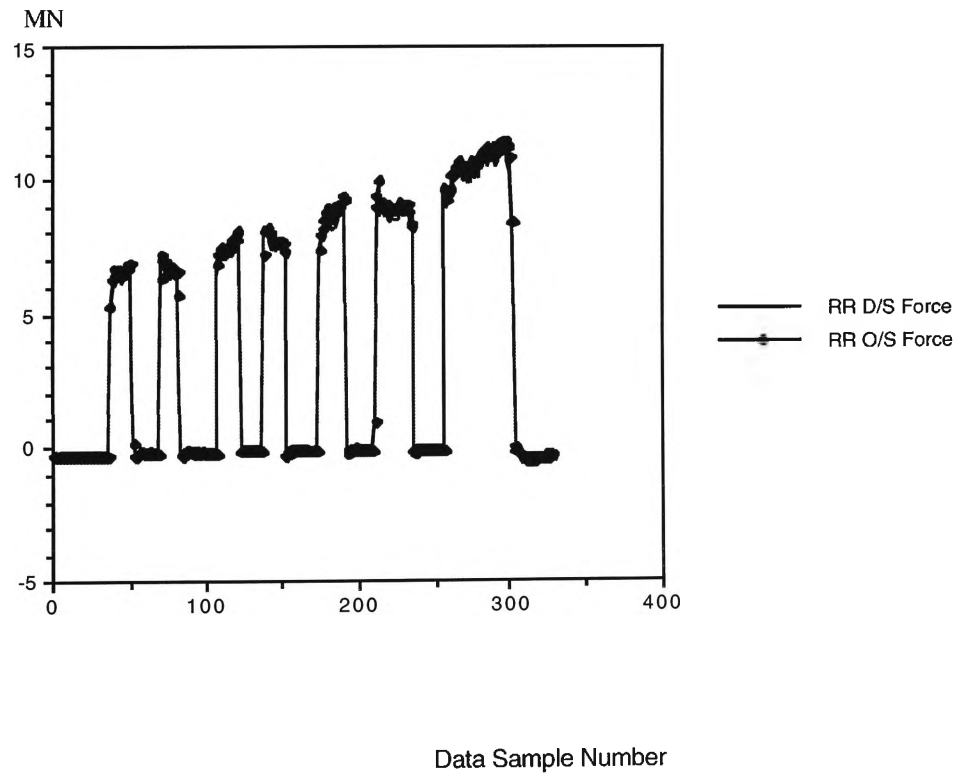


Fig. 6.2 RR D/S and RR O/S Force

The relationship between factor variables and exit width is not clear yet. But some that directly influence the width are considered to be more significant. The gap of rolling edger is inevitably the most influential one. This is easy to understand from rolling process theory. The Entry width variation is also crucial for Exit Width Deviation. Other variables that also affect width change will be considered.

From the above discussion of rolling variables, it is clear that there is a redundancy among input variables. The key point is to go through statistical method, find a suitable way to remove this redundancy and, at the same time, try to not lose any useful information.

6.3.2 Analysis of regression

The listed signals in table 6.1 are of different level of importance. Before considered into a regression model, these are sequenced by level of importance to output according to process experience (knowledge). The order is given as.
v2 v1 v3 v7 v5 v9 v10 v4 v11 v12 v6 v8

This may not be an accurate sequence as this is based somehow on intuition.

1. Step-wise regression analysis

The step-wise regression model is a modification of the forward-selection technique and differs in that variables already in the model do not necessarily stay there. The model begins with no variables in the model. For each of the independent variables, FORWARD calculates F statistics that reflect the variable's contribution to the model if it is included. The p values for these F statistics are compared to the SLENTRY= value that is specified

in the MODEL statement. After a variable is added, the stepwise method looks at all the variables already included in the model and deletes any variable that does not produce a statistically significant F test at the SLSTAY= level. Only after this check is made and the necessary deletions accomplished can another variable be added to the model. If none of the variables outside the model has an F statistically significant at SLENTRY= level and every variable in the model is significant at the SLSTAY= level, or when the variable to be added to the model is the one just deleted from it, the STEPWISE stops.

The following Panel 6.1 is the SAS program for carrying out the analysis.

Panel 6.1 SAS program for stepwise regression

```
data pass;
infile 'c:\p.nna';
array vr{13} v1-v13;
input id vr{*};
run;

proc reg data=pass;
model v13= v2 v1 v3 v7 v5 v9 v10 v4 v11 v12 v6 v8
/ selection=stepwise include=2
output out=l p=pl r=rl;
data all; merge l;
run;

goptions reset=global gunit=pct device=win cback=white
ftext=centxi htext=3;
title1 height=6 'Regression Analysis';
title2 height=4 'v13 = stepwise selection ';
axis1 minor=none value=(font=swissl) offset=(1,22)
label=('position') width=1;
axis2 minor=none value=(font=swissl) offset=(0,0)
label=('') width=1;
symbol1 interpol=spline width=1 color=red;
symbol2 interpol=spline width=1 color=blue line=2;

proc gplot data=all;
plot v13*id=1 pl*id=2/ overlay frame haxis=axis1 vaxis=axis2;
run;
```

The result is the selection in the order as they enter the model. It is also shown in Panel 6.2.

v11 v5 v1 v6 v12 v10 v9 v7 v2 v8 v3 v4

Panel 6.2 Stepwise regression result

Step	Variable Entered	Number In	Partial R**2	Model R**2	C(p)	F	Prob>F
1	V11	1	0.0458	0.0458	1246.3904	104.4120	0.0001
2	V5	2	0.1117	0.1576	848.1231	288.1860	0.0001
3	V1	3	0.0136	0.1712	801.2997	35.7136	0.0001
4	V6	4	0.0726	0.2438	543.0512	208.5600	0.0001
5	V12	5	0.0216	0.2654	467.6963	63.7840	0.0001
6	V10	6	0.0052	0.2706	451.2255	15.3309	0.0001
7	V9	7	0.0699	0.3405	202.7874	229.7921	0.0001
8	V7	8	0.0270	0.3675	107.9724	92.5863	0.0001
9	V2	9	0.0124	0.3799	65.3780	43.4827	0.0001
10	V8	10	0.0107	0.3907	28.8799	38.1827	0.0001
11	V3	11	0.0045	0.3952	14.7736	16.0857	0.0001
12	V4	12	0.0011	0.3962	13.0000	3.7736	0.0522

Different result may be produced by selecting different Entry level and Stay level. By default, they are set to 0.15. If we set SLENTRY=0.01 and SLSTAY=0.005, v4 is then removed from stepwise regression model.

2. Grouped stepwise regression

Another option for doing stepwise is to draw a line between variables and to form groups. For variables of the group, these are added or delete at the same time.

Panel 6.3 Grouped stepwise regression

```
proc reg data=pass;
model v13=v2 v1 v7 {v3 v5} {v9 v10} v4 {v11 v12} v6 v8
/ selection=stepwise sle=0.01 sls=0.005 include=2
 groupname= 'v2' 'v1' 'v7' '{v3 v5}'
'{v9 v10}' 'v4' '{v11 v12}' 'v6' 'v8';

output out=l p=pl r=rl;
```

```
data all; merge l;
run;
```

The result is, with group { v9 v10} { v11 v12}, the selection is the following variables in the order as they enter the model, {v9 10} v1 v6 v7 v5 v2 {v11 v12} v8 v3.

The R-square for these models are too low as it is in the 0.39 level. This indicates that some interaction between variables may exist. It may be considered a high order form of the variables, nonlinear function of the variables and multiplication among them.

If there is an obvious trend in *residual plot*, some items may be needed. Graphic plot of the variables against residual are printed out and studied. The plots, indicate that there is no specific nonlinear function trend. The residual seems to be independent. Fig. 6.3 is one of the residual plot.

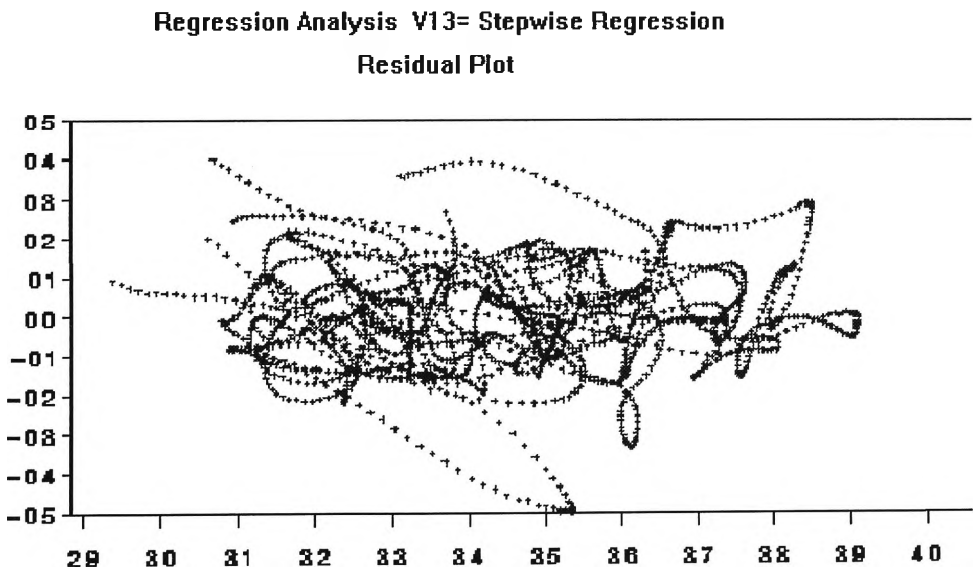


Fig. 6.3 Residual Plot of Variable 7

6.4 Principle Component Analysis

In statistical analysis, there exists a method called Principle Component Analysis.(Panel 6.4) which can be used to confirm the result from step-wise regression. The Principle Component Analysis tries to explain the variance-covariance structure through a few linear combinations of the original variables. It is generally used in data reduction and model interpretation. Both objectives are of our concern.

With a system, which has p components, it still can be accounted for by a small number, k, of the principal components. In such case, there should be almost as much information stored in the k components as there is in the original p variables. This k components usually are combination of original p variables.

In this method, the covariance matrix for input variables is calculated. Algebraically, principal components are particular linear combination of the p random variables $X_1, X_2, ..., X_p$. Geometrically, these linear combinations represent the selection of new coordinate axes. The new axes represent the directions with maximum variability and provide a simpler and closer description of the covariance structure.

Suppose the random vector $\mathbf{X}'=[X_1, X_2, ..., X_p]$ have the covariance matrix $\sum$ with eigenvalues $\lambda_1 \geq \lambda_2 \geq ... \lambda_p \geq 0$. The linear combinations of X can be written as:

$$\begin{aligned} Y_1 &= \theta'_1 \mathbf{X} = \theta_{11}X_1 + \theta_{21}X_2 + ... + \theta_{p1}X_p \\ Y_2 &= \theta'_2 \mathbf{X} = \theta_{12}X_1 + \theta_{22}X_2 + ... + \theta_{p2}X_p \\ &\vdots \qquad\qquad \vdots \\ Y_p &= \theta'_p \mathbf{X} = \theta_{1p}X_1 + \theta_{2p}X_2 + ... + \theta_{pp}X_p \end{aligned} \tag{6-16}$$

The principle component are those uncorrelated linear combinations $Y_1, Y_2, ..., Y_p$, whose variance are as large as possible. The objective of principle component is to construct uncorrelated linear combinations of the measured characteristics that account for much of the variation in the sample.

Below are the principle component analysis carried out in SAS for windows environment.

Panel 6.4 Principle component analysis

```
proc princomp cov data=pass out=result;
var vr1-vr13;
run;
```

Panel 6.5 Principle component analysis result

Principal Component Analysis

Eigenvalues of the Covariance Matrix

	Eigenvalue	Difference	Proportion	Cumulative
PRIN1	0.453290	0.285320	0.638386	0.63839
PRIN2	0.167971	0.116703	0.236560	0.87495
PRIN3	0.051268	0.029498	0.072202	0.94715
PRIN4	0.021769	0.013527	0.030659	0.97781
PRIN5	0.008242	0.005448	0.011608	0.98941
PRIN6	0.002794	0.001034	0.003935	0.99335
PRIN7	0.001760	0.000435	0.002479	0.99583
PRIN8	0.001325	0.000636	0.001866	0.99769
PRIN9	0.000690	0.000172	0.000971	0.99867
PRIN10	0.000518	0.000242	0.000730	0.99940
PRIN11	0.000276	0.000122	0.000388	0.99978
PRIN12	0.000154	.	0.000217	1.00000

Principal Component Analysis

Eigenvectors

	PRIN1	PRIN2	PRIN3	PRIN4	PRIN5	PRIN6
V1	-.098914	0.952739	-.122607	-.208574	-.118236	0.020605
V2	0.049892	-.057096	-.012800	-.239582	0.008982	0.219379

V3	-.111770	0.151555	0.072042	0.264528	0.576130	-.650786
V4	-.057074	0.041060	0.029497	-.057128	0.154707	0.279117
V5	-.159750	0.155779	0.187640	0.854452	-.308893	0.214885
V6	-.001123	0.063806	0.038469	-.071467	-.013014	0.180865
V7	-.015025	-.032608	-.888309	0.247719	0.290969	0.237525
V8	0.968062	0.155345	0.052232	0.164900	0.081660	0.018258
V9	-.060488	0.053314	0.256692	0.011383	0.467297	0.410350
V10	-.059164	0.068896	0.256654	-.003955	0.410830	0.300222
V11	-.028703	0.014350	0.131307	0.044072	0.156956	0.212934
V12	-.033205	0.055182	0.035076	0.004372	0.177144	-.102562

	PRIN7	PRIN8	PRIN9	PRIN10	PRIN11	PRIN12
V1	-.007020	0.000370	-.088554	0.021141	-.011889	-.032937
V2	0.930620	0.037956	-.086093	-.028438	-.010584	-.112485
V3	0.225895	0.253405	0.062556	-.089941	0.081731	0.000227
V4	-.092263	0.344605	0.582449	-.306626	-.459069	-.343231
V5	0.193034	-.044436	0.028669	-.048340	-.020590	-.022505
V6	0.060198	0.123996	0.581039	-.032257	0.576953	0.515692
V7	-.002457	-.028682	-.038731	0.029208	0.018733	0.060325
V8	-.004025	0.007783	0.030190	-.011203	-.022600	-.013331
V9	-.150134	-.181544	-.205113	-.159020	0.489704	-.421969
V10	-.011452	-.135846	-.221936	-.111608	-.429419	0.634235
V11	-.037529	0.437059	-.028697	0.844145	-.035469	-.052136
V12	0.096792	-.744934	0.456723	0.376769	-.150646	-.123141

The result of Principle Component suggests that 5 principle components will take up 99% of output. After studying the 5 principle component, v1 v3 v5 v7 shows strong contribution to principle component. In consideration of this, we add the interaction (multiplication) items of these 4 variables into the regression model.

v13 = 100.117+78.446*v1+0.408*v2-25.843*v3+0.520*v4-98.566*v5-2.984*v6-30.492*v7+0.017*v8+2.543*v9-2.943*v10-0.907*v11-0.279*v12
+0.290*v1*v1-25.250*v1*v3-79.845*v1*v5-26.18*v1*v7-1.768*v3*v3
+9.450*v3*v5+35.627*v3*v7-1.599*v5*v5+26.444*v5*v7+0.493*v7*v7
+25.692*v1*v3*v5+8.568*v1*v3*v7+26.650*v1*v5*v7-9.267*v3*v5*v7-8.687*v1*v3*v5*v7

(6-17)

From this statistical model, some suggestion can be made on how to improve on the neural network, which is to be discussed in the next chapter. It is therefore suggested that v4 be disregarded in the model while add in some interaction into neural networks model. All the example analysis in this chapter is carried on the pass 3 data. Similar results were obtained when working on other pass data set.

Chapter 7

Neural Network Width Model Optimization and Analysis for Practical Use

After the statistical analysis of model variables, the neural network model is subjected to change in its structure. At the same time, other requirements from the production units force some modification of the neural network, so as to make it suitable for practical application. These changes are discussed and tried in this chapter. Also presented is the analysis of the final neural network model. It is carried out to have a better understanding on all factors contributing to the exit width deviation.

7.1 Practical requirement for the Neural Network Model

The neural network models discussed in previous chapters are solely based on input-output signal relationships. The structures are simple without any consideration of other factors. However, if the industry wants to apply the models into production, it would have to satisfy some specific requirements for such application.

As stated in Chapter 5, the width models initially developed are pass-based models, which take into account slabs of the same grade and same section (same aimed width and same aimed thickness). Each section within a grade of slabs has its own model. This would mean that for every new section of a grade, there should be a new network model corresponding to it. An application program would have to include all the possible models for use. It, in turn, would make the database for width models quite large, therefore is not suitable for practical applications. It also will create a problem, if there is a new section to be rolled for a known grade, there may not be an exact model which can match it.

The practical requirement for model application is, that on the premise of good performance of neural network models, the number of models should be as few as possible. Or, the model should have some flexibility so that it may be applied to a wide range of slabs (e.g. model for a whole grade). Also, the model should be able to cope with novelty situation and have good interpolation results.

7.2 The modification of the in-bar width model

Modifications to the neural network in-bar width model are then made according to the requirements from the above practical application viewpoint and the statistical analysis.

7.2.1. Modification according to statistical analysis

In the suggestion from chapter 6, it says that variable 4, RE speed, can be discarded without significantly affecting the result accuracy. At the same time, some interaction items can be added into the model. This is considered from the statistical point of view. The suggested interaction items are those variables Entry width v1, RE force v3 , RE current v5 and RR Exit temperature v7.

The modification to the neural network then is made according to the above suggestions. The RE speed is left in the model. In addition to the variables in Fig. 5.3, 15 more inputs are included into the model. They are the interactive forms, v1*v1, v1*v3, v1*v5, v1*v7, v3*v3, v3*v5, v3*v7, v5*v5, v5*v7, v7*v7, v1*v3*v5, v1*v3*v7, v1*v5*v7, v3*v5*v7, v1*v3*v5*v7. This is not a full rank interaction. There are more combinations with these four variables.

7.2.2 Modification regarding practical requirement

The main idea of practical requirement is to add some flexibility to the neural network model, to allow the model based at least on grade level rather than sections of a grade. It

is acceptable that the network is still pass-based. From this understanding, a solution to make one model (of one pass) for a whole grade of slabs rolled is desired.

A possible solution to fulfil the requirement is to add some more feature variables to let the network model suit for the requirement. Those feature variables would have to take into account the variation from one section to another within the particular grade of concern. The different considerations between two sections are the aimed width and the aimed thickness. The corresponding variables in the setup information for roughing mill are the aimed width and RR Gap. The RR gap setup value would have the same effect as the aimed thickness as there is little difference between them for every pass.

Another factor is that the roughing mill edger would have different width reduction depending on the section. The width reduction is the next variable to enter the model. It is also a setup value from the computer. It only changes from one bar to another. For slabs in the same section, it is almost the same. This means that this input in neural network model would contribute to section change.

After considering the section change within a grade of steel, three setup variables are selected to add more flexibility to the model, which are then grade based. The number of grades for the strip product is relatively small. This solution would greatly reduce the space required for any application in-bar width neural network model database. The database can be used in either mill control computer or in any expert system.

7.2.3 The final neural network model structure

The final neural network model after modification is shown in Fig. 7.1, which includes the modifications made in section 7.2.1 and 7.2.2. The interaction items are those from statistical consideration. The three setup variables, the Aimed Width, the RR Gap and the RE Width Reduction, are from application requirement.

To use the new model structure, the training and testing data set has to be re-organized to add all these feature variables and interaction of variables. The number of hidden layer neuron also has to be modified according Eq. (5-1).

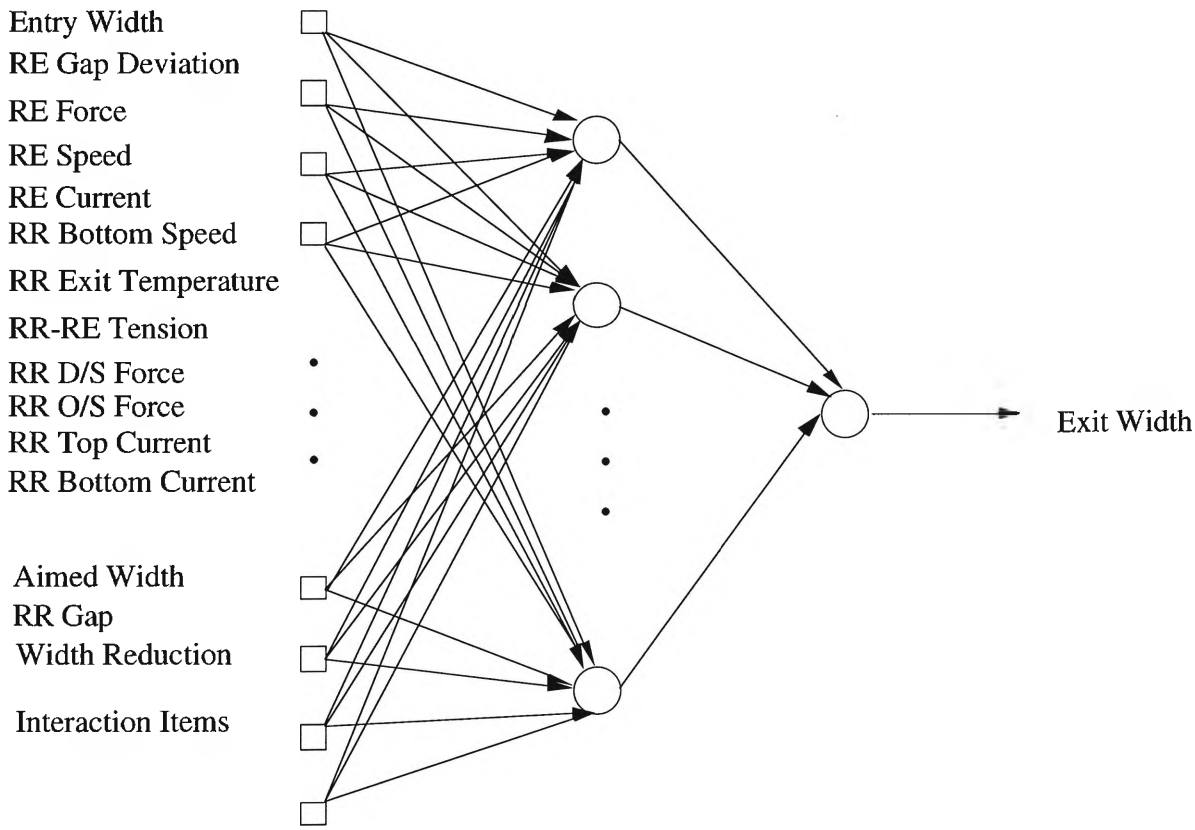


Fig. 7.1 Modified Neural Network model for in-bar width

7.3 Result and interpretation of the modified neural network model for in-bar width

7.3.1 Some result of the modified neural network model.

After modification of the network structure, we tried the network on a new data set. The following result is obtained from the network trained on plain carbon manganese grade. The data set are slabs with dimensions (Width × Thickness) of 1129×25.0, 1108 × 25.0, 908 ×25.0 and 907 × 25.0. The coil number of slabs used for training are I6152, I6154, I6155, I6157, I6165, I6167, I6171, I6172, I6173, I6175, I6176 and I6178. The ones used on testing are I6153, I6156, I6174 and I6179.

The testing results of the network model are shown below. Fig. 7.2 is the result for pass 3, Fig. 7.3 is the one for pass 5, Fig. 7.4 is the one for pass 7. From the graphs, it can be seen that the neural network really have a good interpolation effect of the data. t-test and F-test results are also shown below.

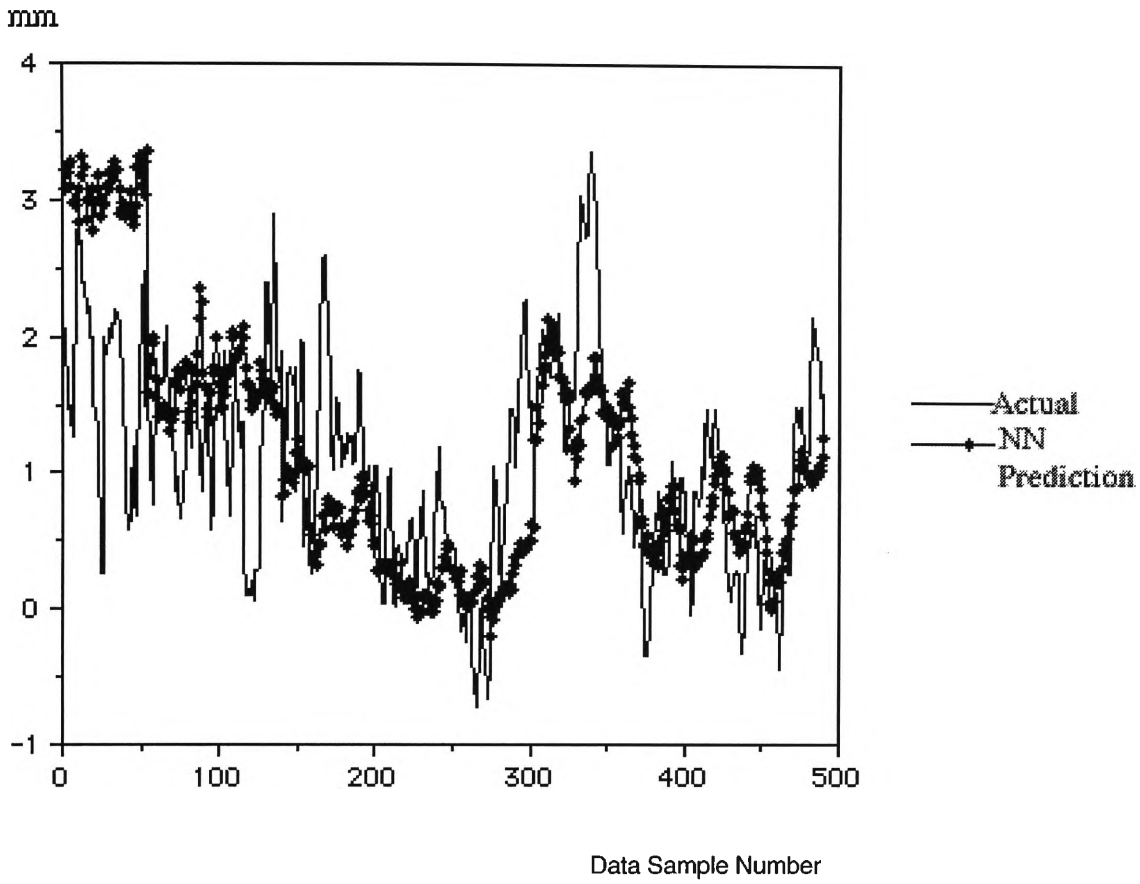


Fig. 7.2 Result of pass 3 modified width model

Pass3: Measured data, Mean =1.080 mm, Std dev=0.771mm;

NN Predicted, Mean=1.005 mm, Std dev= 0.864mm.

NN Predicted- Measured, Mean=0.074mm, Std dev=0.763 mm

t-test, (Aimed-Measured) Z=31.008, (Predicted-Measured) Z=2.147,

F-test, F= 1.021

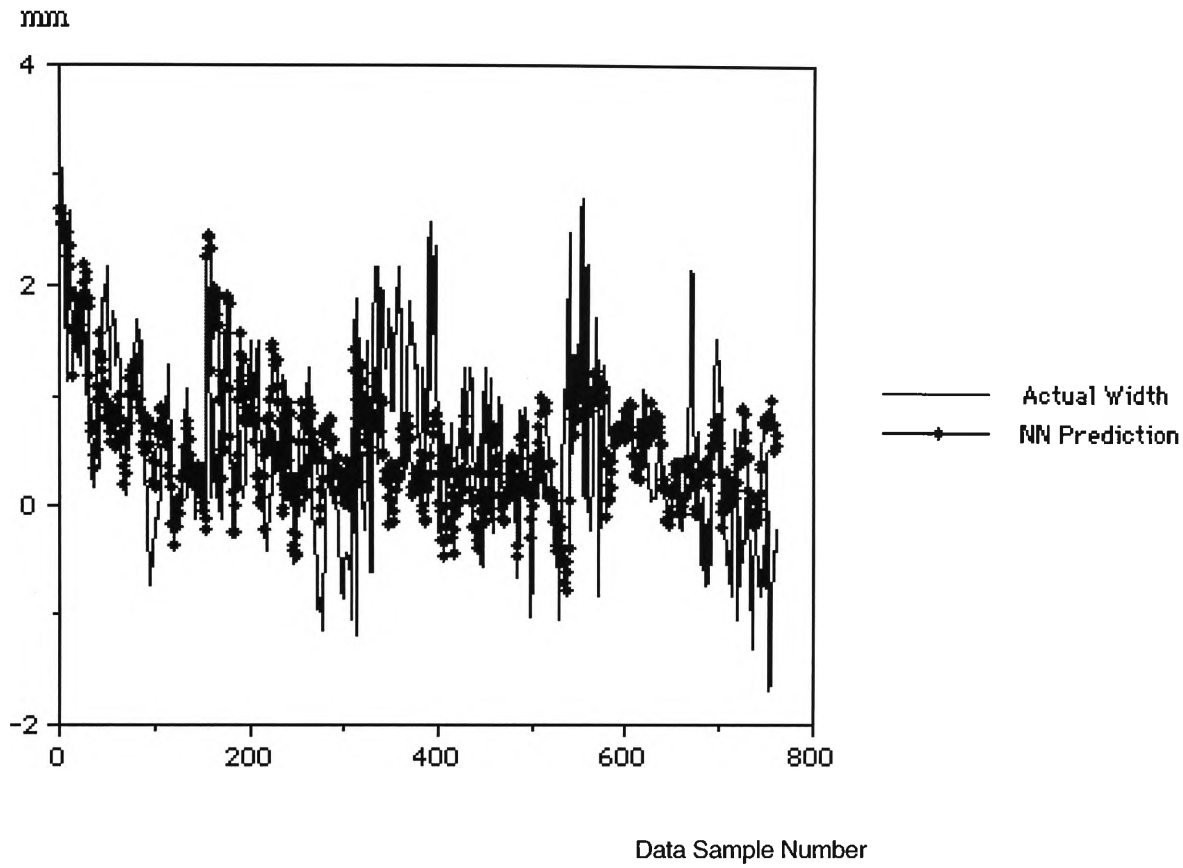


Fig. 7.3 Result of modified pass 5 width model

Pass5: Measured data, Mean =0.525 mm, Std dev=0.815 mm;

NN Predicted, Mean=0.560 mm, Std dev= 0.552 mm.

NN predicted- Measured, Mean=0.035 mm, Std dev=0.807 mm

t-test, (Aimed-Measured) Z=17.875, (Predicted-Measured) Z=1.203

F-test, F=1.020

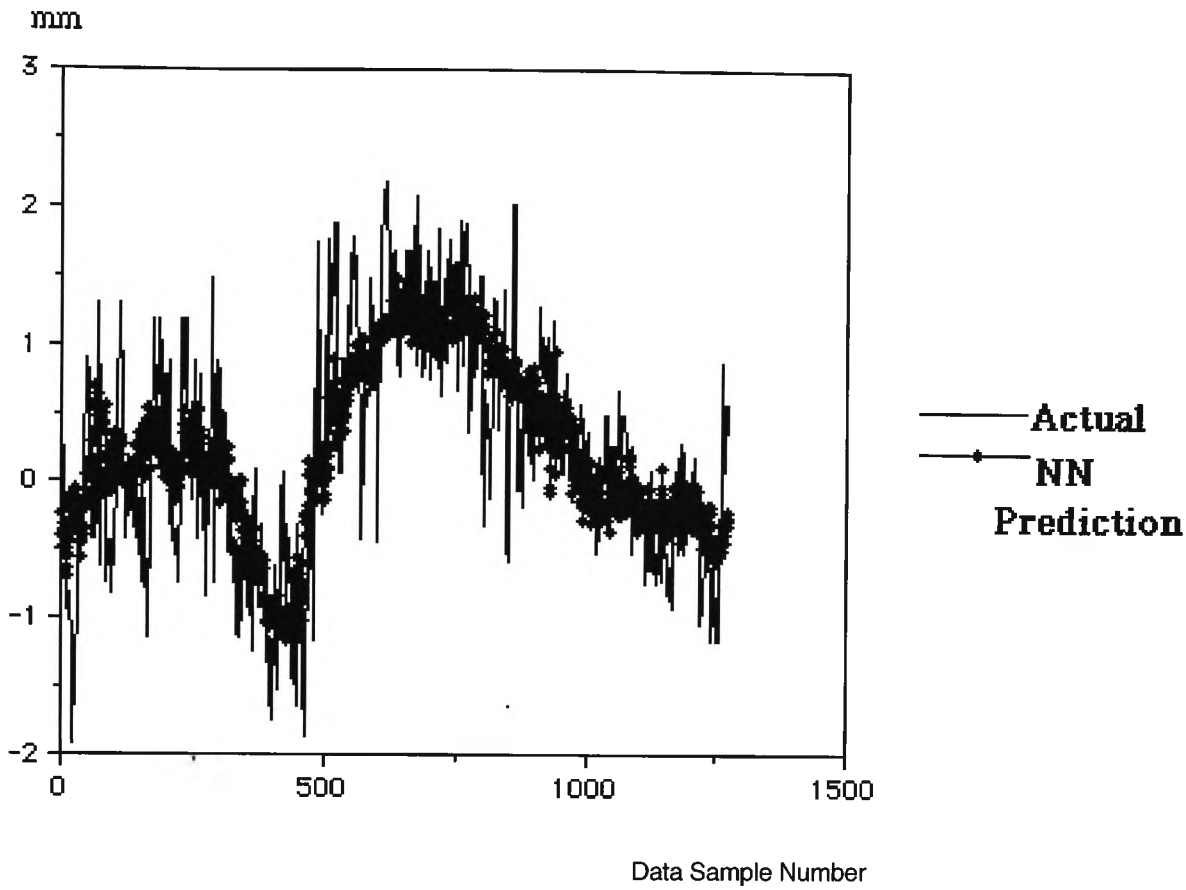


Fig. 7.4 Result of pass 7 modified width model

Pass7: Measured data, Mean =0.235 mm, Std dev=0.816mm;

NN Predicted, Mean=0.223 mm, Std dev= 0.64mm.

NN Predicted- Measured, Mean=0.012mm, Std dev=0.509 mm

t-test, (Aimed-Measured) Z=10.384, (Predicted-Measured) Z=0.850

F-test, F=2.570

7.3.2 More results with another plain carbon manganese grade slab model

Data are collected with slabs of another plain carbon manganese grade (Grade 2). Since before this, pass 1 was not included in the model. The reason was the noise with width

gauge 1. Here, some of the slabs, WG0 signal is used as WG1 signal for Neural Network Training.

(i) Pass One: Trained on 42 coils, which have 4 sections of width:

1167, 1150, 1268 and 1238.

The neural network model is tested on four bars: D1303, D1305, D1306, D1307.

The graph is shown in Fig.7.5. The solid line is measured data. Dashed line is the neural network prediction.

Measured: Mean=2.290 mm, Std dev=0.916 mm

Predicted: Mean=2.020 mm, Std dev=0.925 mm

Error: Mean=0.27 mm, Std dev=0.864 mm

t-test, (Aimed-Measured) Z= 55.340, (Predicted-Measured) Z=5.636

F-test, F= 1.124

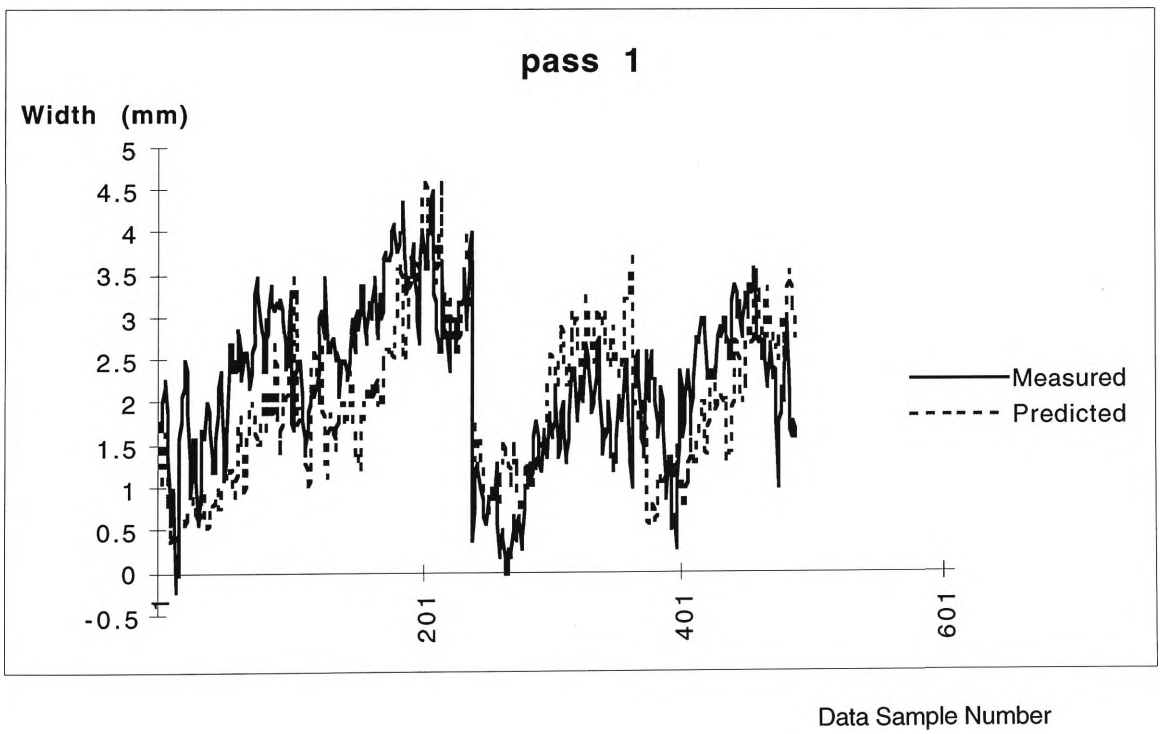


Fig. 7.5 Pass 1, A06 Width model

(ii) Pass 3 model

Trained with 90 coils, with 8 sections(Different aimed roughing bar width size):

980, 1019, 1030/1029, 1150, 1157, 1166/1167, 1237 , 1268.

The model is tested on different sections. Fig.7.6 show test results on bar D3521, D3523, D3524, D3527, D3556, D3558, D3559.

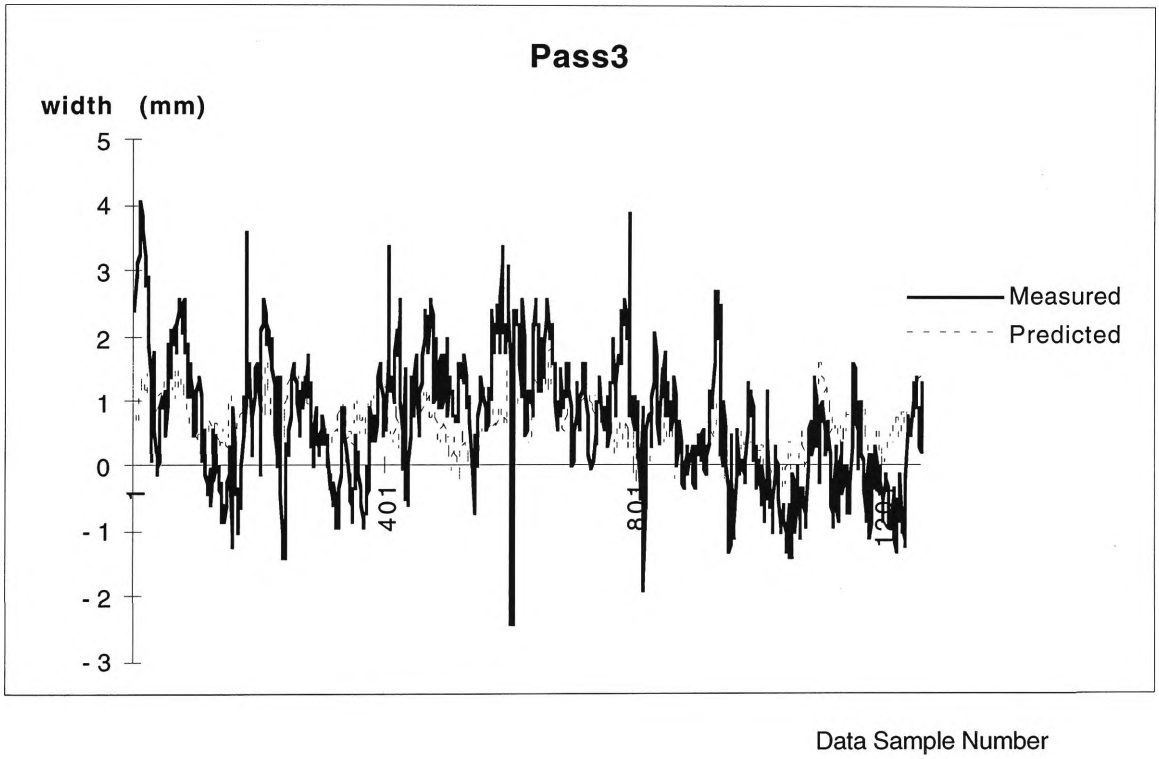


Fig. 7.6 Pass3 test result on A06 model

Statistics of the test result:

Measured: Mean=0.721 mm, Std dev=0.998 mm

Predicted: Mean=0.660 mm, Std dev=0.442 mm

Error: Mean=0.061 mm, Std dev=0.879 mm

t-test, (Aimed-Measured) Z= 25.542, (Predicted-Measured) Z=2.454

F-test, F= 1.289

(iii) Pass 5.

Trained with 90 coils, with 8 sections(Different aimed roughing bar width size):

980, 1019, 1030/1029, 1150, 1157, 1166/1167, 1237 , 1268.

The model is tested on different sections. Fig.7.7 shows test results on bar D3521, D3523, D3524, D3527, D3556, D3558, D3559. The statistics of the results are as follows.

Measured: Mean=0.154 mm, Std dev=1.227 mm

Predicted: Mean=0.186 mm, Std dev=0.940 mm

Error: Mean=0.032 mm, Std dev=0.838 mm

t-test, (Aimed-Measured) Z= 5.02, (Predicted-Measured) Z=1.527

F-test, F=2.144

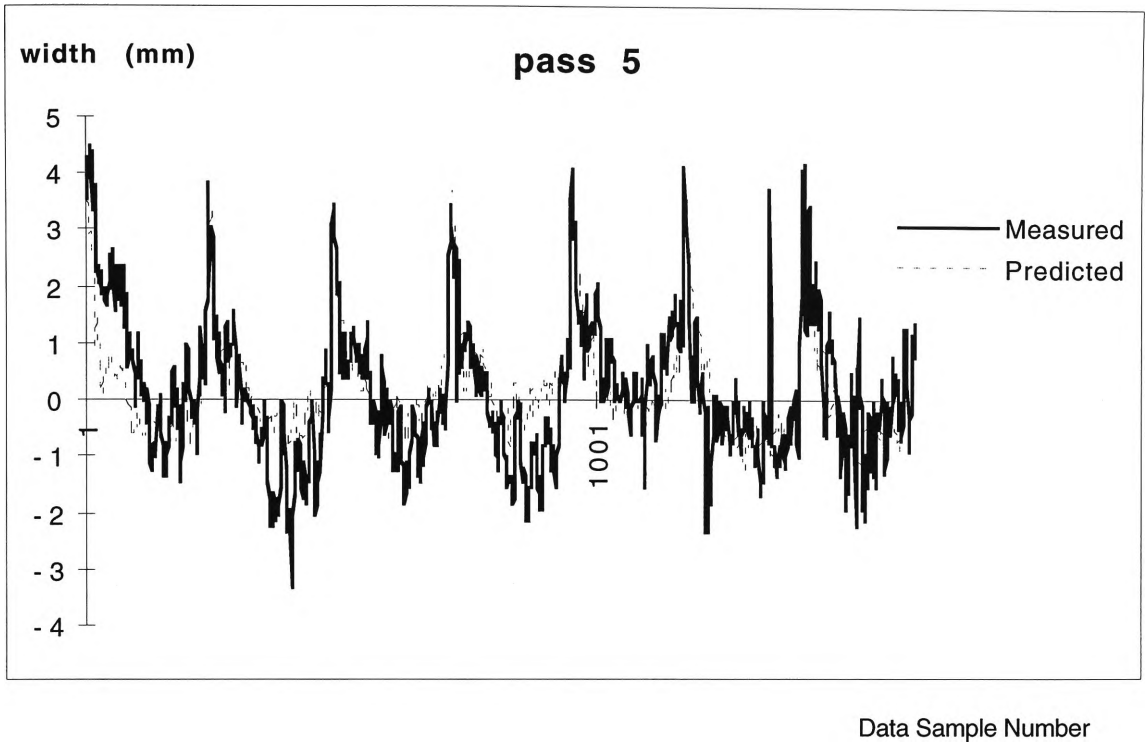


Fig. 7.7 Pass 5 test result on A06 on NN width model

(iv) Pass 7 model

Trained with 90 coils, with 8 sections(Different aimed roughing bar width size):

980, 1019, 1030/1029, 1150, 1157, 1166/1167, 1237 , 1268.

The model is tested on different sections. Fig.7.8 shows test results on bar D3521, D3523, D3524, D3527, D3556, D3558, D3559. The statistics of the results are as follows.

Measured: Mean=0.309 mm, Std dev=0.750 mm

Predicted: Mean=0.167 mm, Std dev=0.515 mm

Error: Mean=0.142 mm, Std dev=0.655 mm

t-test, (Aimed-Measured) Z=24.72, (Predicted-Measured) Z=13.099

F-test, F=1.311

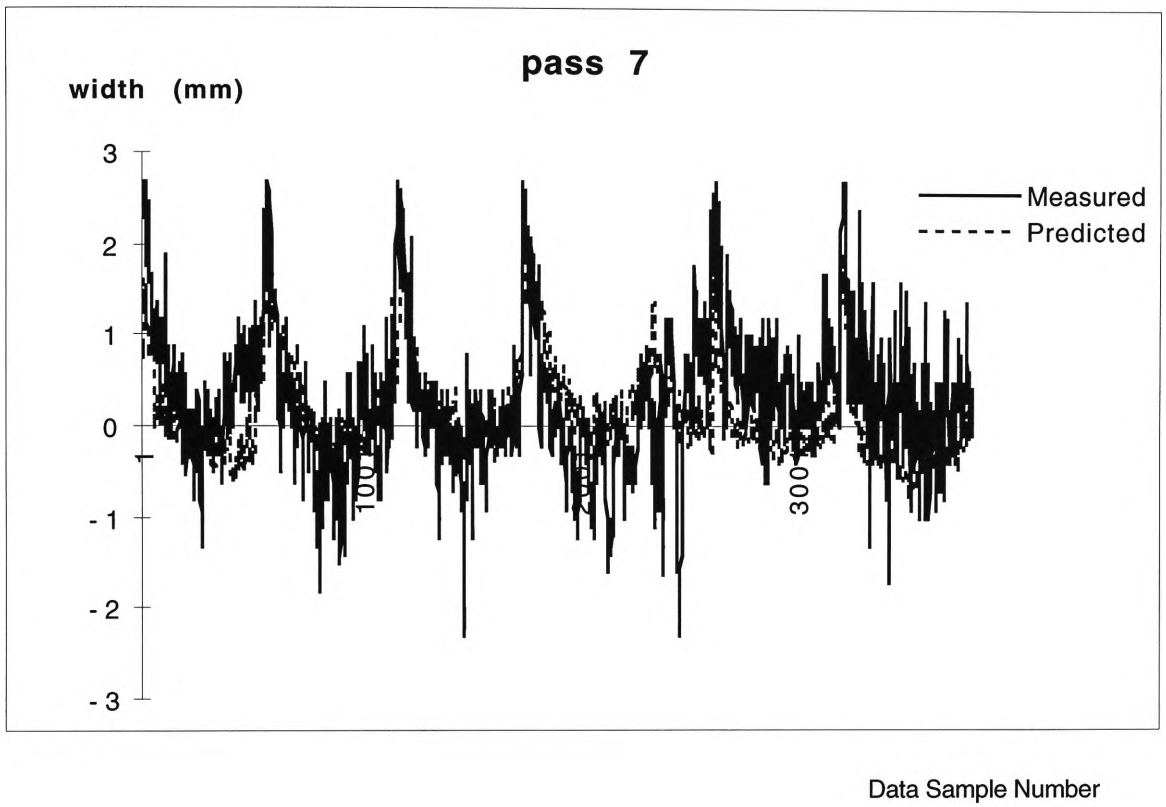


Fig. 7.8 Pass 7 test result on A06 on NN width model

These test results show good interpolation of the measured data. At some bars, the error of prediction becomes noticeable due to the oscillation of original signals, which passed the filter.

7.3.3 Interpretation of the modified neural network model

In chapter 6, we did stepwise regression to analyze the importance of all the network variables. It is sequenced according to each variable's statistical value to the model. There is another way to interpret the model, which is based on the weights of the

network. This kind of method is called dithering. This method allows the network input to dither a certain percentage and watch the effect at the output. This may give an indication of how much each variable influences the output.

The following graphs are the results from the 5% dithering of the modified models trained in the above section. Fig 7.9 , 7.10, 7.11 are results of interpretation for pass 3, pass 5, pass 7 respectively.

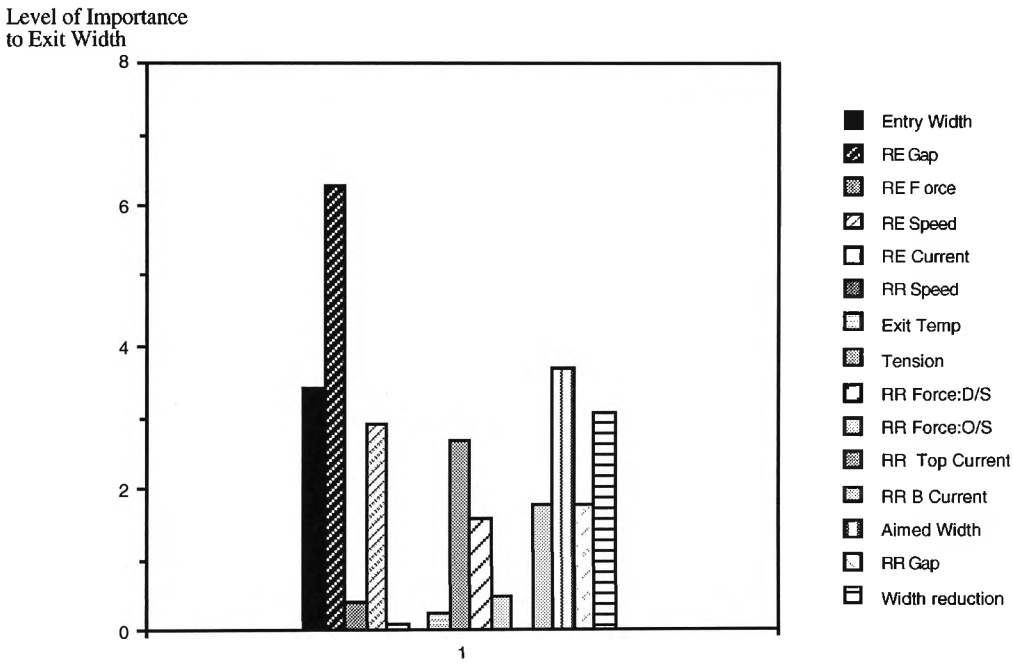


Fig. 7.9 Analysis of variables for pass 3 model

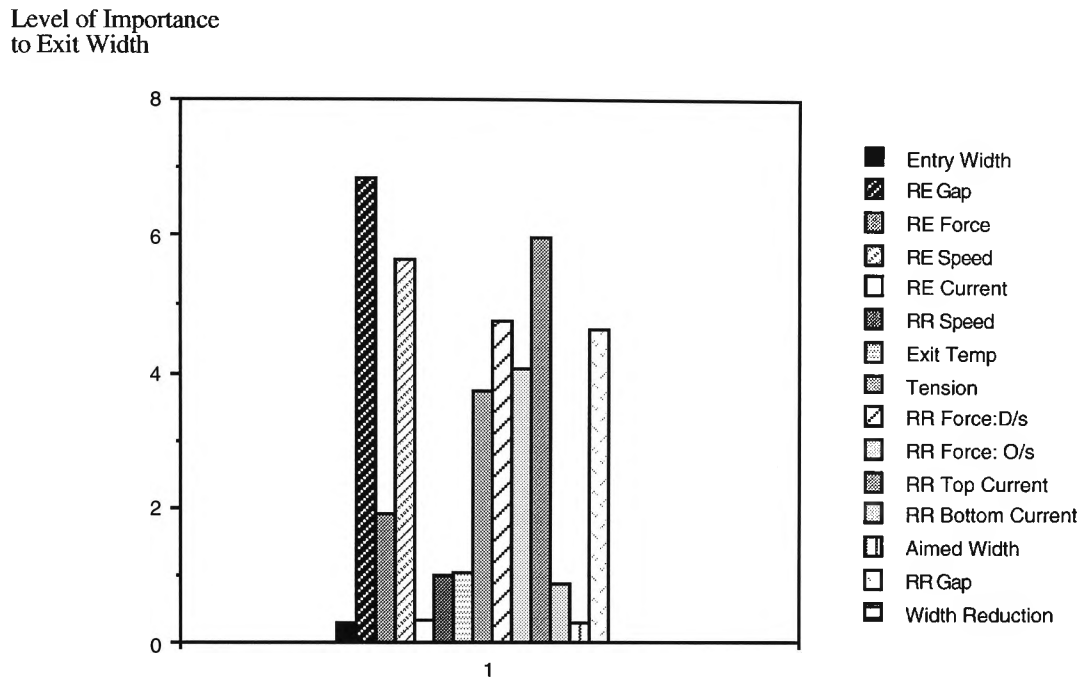


Fig. 7.10 Analysis of variables for pass 5 model

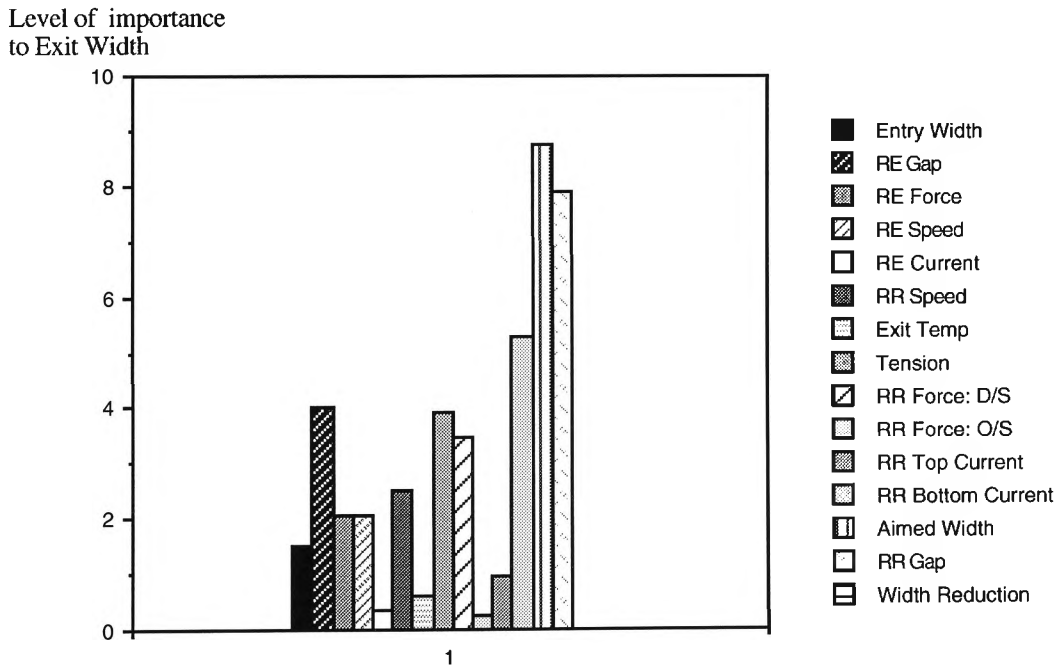


Fig. 7.11 Analysis of variables for pass 7 model

From the graphs, it can be seen that RE Gap always play an important role in width change. This is the most easily understood from the rolling theory. Other variables, the importance is different with different pass. This can be taken as a comparison to the statistical analysis in Chapter 6. Table 7.1 shows the comparison level of importance from dithering and statistics for pass 3 with regarding to the width deviation.

Table 7.1 Comparison of variable importance to width deviation

Method\order	1	2	3	4	...
Dithering	RE Gap	Aimed Width	Entry Width	Width reduction	...
Statistics	Entry Width	RE force	RE Current	RR Exit Temp.	...

Note that in dithering model, there are mill set-up variables, which don't appear in statistics model. It is especially useful for the suggestion of control strategy for in-bar width control, to suggest adjustment to some variables, which needs further research.

Chapter 8

Conclusions and Recommendations

The Neural Network Model for In-bar Width has been described in previous chapters. From the procedures and background information, some conclusions are able to be drawn.

The Neural Network model is suitable for width prediction. The results obtained so far have proven that the Neural Network can model the width change process in the Roughing Mill. In Chapter 5, a neural network model was built, then through to Chapter 7, it was analysed and modified to suit the need for modelling and practical use.

The prediction results, compared with the actual results from the mill, show fairly good interpolation of the actual results. Considering that there are noise signals in actual results, the prediction is more than acceptable. It also filters the noise signals.

The neural network model shows an ability to fit data with novelty, which is not available with any other width models. Empirical models and statistical (regression) models for width change are inflexible in many situations. In the model in Chapter 7, some variables had been added to the model in order to use the model flexibly with different coil schedule.

The neural network is a versatile tool for modelling and prediction. It offers nonlinear feature, good interpolation ability and adaptability to novelty situation.

The neural network is simple to understand and easy to use. It requires no specific process information to understand the complex physical and/or chemical change happened. The network provides the ability to learn from the environment just like human being. To use the network, one only needs to supply the required input signal. The network works like a black box, users do not need to worry about what is inside the black box. However, this does not mean that physical knowledge about the process being modelled is useless. Expertise on the process or a good understanding of the physical process would give light on improving network model accuracy.

To design a neural network application, it is very important to try different network parameters in order to get an optimal network model. The proper way to build up a network is by trial and error. There is no formulated procedure to follow.

Specific information about the process to be modelled by neural network can shorten the time to get desired network. For example, rolling gap is the mean for width control, so it should be kept in the model, no matter whatever result may be obtained in statistical analysis. Although not required for neural network modelling, this information may avoid a misleading network since the first moment of construction of the network and the statistical analysis as well.

Statistical analysis can help greatly in improving model accuracy and reduce network redundancy. Work in Chapter 6 is the most common way to analyse, although other statistical methods are available.

A neural network model has to satisfy certain industrial application requirements. Such requirements would make model more practically acceptable, rather than just used in academic research. The modification to fulfil these requirements makes the model flexible and sometime improves accuracy as well.

The pre-processing of neural network is very crucial for the use of network model. The pre-processing gives a neural network the possibility to model a physical process.

Without proper pre-processing, there will be no correct information to train the network. This is particularly important in this application, all the data need to be properly synchronised and normalised, as in stated in Chapter 4.

Since the purpose of the neural network modelling is to supply an accurate prediction model for mill control, further study can be carried out on how to use the prediction results in in-bar width control. As the mill width control is conducted by adjusting roughing mill edger gap, the question is then how to control the gap so as to accurately control the width. Fuzzy logic is a booming control algorithm , which can be used in adjusting gap. Neural network adaptive control is another possible way to do the width control. All those need further study and test on the mill or in simulation machine.

Publication

Luo, S. Z., Tieu, A.K. and Fang, X. D., A Fuzzy Featuring Algorithm for Monitoring the Rolling Process, Proceeedings of Second Annual Joint Conference on Information Sciences, NC, 1995., pp. 494-497.

References

Aleksander, I., and Morton, H, An Introduction to Neural Computing. London:Chapman & Hall, 1990.

Anon, Intelligent arc furnace operation at Birmingham steel, Steel Times International, v 20, n 1, Jan 1996, pp. 20-21.

Auzinger, D., Pfaffermayr, M., Pichler, R., Schlegl, B., Advanced process models for today's hot strip mills, Metallurgical Plant and Technology International, v 18, n 6, Dec 1995, 6pp.

BHPE Manuals & Documentation (Port Kembla), BHP SPPD Hot Strip Mill.

Buchanan, B. G., Sutherland, G. L., and Feigenbaum, E. A.(1969). Heuristic DENDRAL: a program for generating explanatory hypotheses in organic chemistry. *Machine Intelligence 4*, pp. 209-254. Edinburgh University Press, Edinburgh, Scotland.

Bulsari, A., Saxen, H., Steel Research, Classification of blast furnace probe temperatures using neural networks, v 66, n 6, Jun, 1995, pp. 231-236.

Burras, S., Real-time expert systems in the steel industry, Steel Times, v 223, n 4, Apr 1995,. 2pp

Burras, S., Real time expert systems: delivering real benefits, Materials World, v 3, n 4, Apr 1995, pp. 173-175.

Charniak, E. and McDermott, D. (1985). *Introduction to Artificial Intelligence.* Addison-Wesley, Reding, Massachusetts.

Cohen, J. M., Lessons learned implementing expert systems technology within the steel industry, Iron & Steelmaker, V 21, n 4, Apr., 1994, pp. 31- 35.

Dahl, W., Microstructure and mechanical properties, Stahl and Eisen, March 19, 1884, pp. 29- 36.

Desrochers, A. A, Saridis, G. N., Control methods for a Hot Steel Rolling Mill: An application of learning theory and pattern recognition, Transactions of the ASME Journal of Engineering for Industry, V. 102, May, 1980, pp.118-122.

Dorn, J., Expert systems in the steel industry, IEEE Expert, v 11, n 1, Feb 1996, pp. 18-21

Fischer, C. H. (1985). Expert systems and artificial intelligence, Iron and Steel Engineer, Nov., pp. 58-62.

Gaither, S. A., A design environment for integrating expert system and fuzzy logic in process control, ISA 1991, pp. 1-15.

Hasegawa, A. and Taki, F., Development of fuzzy set theory-based shape control system for cold strip mill, Nippon Steel Technical Report, No. 49, April, 1991, pp. 59-62.

Hattori, S., Nakajima, M., Katayama, Y., Fuzzy control algorithm and neural networks for flatness control of a cold rolling process, Hitachi Review, V41, n1, 1992, pp. 31-38.

Hattori, S., Nakajima, M., Morooka, Y., Application of pattern recognition and control techniques to shape control of rolling mills, Hitachi Review, v42, n4 1993, pp. 165-170.

Haykin, S., Adaptive Filter Theory, 2nd Ed., Prentice- Hall, 1991.

Haykin, S., Neural Networks: A Comprehensive Foundation, p47.

Hebb, D. O. (1949). The Organization of Behavior. Wiley, New York.

Hopfield, J.J. (1982). Neurons with graded response have collective computational properties like those of two-state neurons. Proceedings of the National Academy of Sciences (USA), 79, pp. 2254-2258.

Hwu, Y., Pan, Y., Lenard, J. G., Comparative study of artificial neural networks for the prediction of constitutive behaviour of HSLA and carbon steels, Steel Research, v 67, n 2, Feb 1996, pp. 59-66.

Jacobs, R.A., Increased rate of convergence through learning rate adaptation, Neural Networks, 1, 1988, pp. 295-307.

Jimichi, Y., Hori, A., Ishihara, H. and Odaira, T. (1990). An expert system of automatic slab assignment for Hot Strip Mill, ISIJ Int., Vol. 30 , No. 2, pp. 155-160.

Jimichi, Y., Ishihara, H., Kodaira, T. and Kitano T., A fully automated slab assignment system for a Hot Strip Mill, The Sumitomo Search, No. 50, July, 1992, pp. 36-42.

Jung, J., Im, Y.; Lee-Kwang, H., Development of fuzzy control algorithm for shape control in cold rolling, Journal of Materials Processing Technology, v 48, n 1-4, Jan 15, 1995,. pp. 187-195.

Konishi, M., Takahashi, T. Omura, K. and Izuka, S., Expert system with simulator for selecting routs on billet conditioning line, KOBELCO Technology Review, n 13, Apr., 1992, pp. 1-4.

Labee, C.J. and Samways, N.L., Developments in the iron and steel industry U. S. and Canada - 1991, Iron and Steel Engineer, Feb., 1992, pp. D-1 to D-34.

Leven, J.; Jonsson, N.G.; Wiklund, O., Artificial neural network for rolling applications, Steel Times, v 223, n 4, Apr 1995,. pp. 137-138.

Lighthill, J. (1973). *Artificial Intelligence: A Paper Symposim.* Science Research Council of Great Britain, London.

Maheral, P.; Ide, K.; Gomi, T.; Pussegoda, N.; Too, J.J.M., Artificial Intelligence Techniques in the Hot Rolling of Steel, Canadian Conference on Electrical and Computer Engineering, v 1, 1995, IEEE, Piscataway, NJ, USA, pp. 507-510.

McCarthy, J. (1963). Programs with common sense. Proceedings of the Symposim on Mechanisation of Though Processes, V1, pp. 77-84, London.

McCulloch, W.S. and Pitts, W. (1943). A logical calculus of the ideas imminent in nervous activity. *Bulletin of Mathematical* Biophysics, 5, pp. 115-137.

Minsky, M. L. and Papert, S. (1969). *Perceptrons: An Introduction to Computational Geometry.* MIT Press, Cambridge, MA, first Edition.

Minsky, M. L. (1975). A framework for representing knowledge. The Psychology of Computer Vision, pp. 211-277. McGrawHill, NY.

Nakamura, N., Nishimura, K., Beppu, Y. and Yoshimine T., An AI System for the Facilities Monitoring On-line System, The Sumitomo Search, No. 50, July, 1992, pp. 68-77.

Oda, T., Satou, N., Yabuta, T. , Adaptive technology for thickness control of finisher set-up on hot strip mill, ISIJ International, v 35, n 1, 1995, pp. 42-49.

Osakada, K., Yang, G., Mori, K., by Combination of Fuzzy Reasoning and Finite Element Simulation, Annals of the CIRP, V.42, 1, 1993, pp. 291-294.

Ouch, C., Sampei, T. and Kozasu, I., The effect of hot rolling condition and chemical composition on the onset temperature of the transformation after hot rolling, Trans. ISIJ, Vol. 221982, pp. 211- 222.

Parker, D. B., Optimal algorithms for adaptive networks: Second order back propagation, second order direct propagation, and second order Hebbian learning, IEEE 1st Int. Conf. on Neural Networks, V2., 1987, pp. 593-600.

Pican, N., Alexandre, F., Bresson, P., Artificial neural networks for the presetting of a steel temper mill, IEEE Expert, v 11, n 1, Feb 1996,. pp. 22-27

Pitsch, N. and Hougardy, H.P., Controlled development of microstructure in steels, Stahl and Eisen, March 19, 1984, pp. 19-25.

Portmann, N. F., Lindhoff, D., Sorgel, G., Gramckow, O., Application of neural networks in rolling mill automation, Iron and Steel Engineer, v 72, n 2, Feb, 1995, pp. 33-36.

Ramon y Cajal, S., 1911, Histologie du system nerveux de l'homme et des vertedres. Paris: Maloine; Edition Francaise Revue: Tome I, 1952; Tome II, 1955; Madrid: Consejo Supreior de Investigaciones Cietificas.

Rao, K.P.; Prasad, Y.K.D.V., Neural network approach to flow stress evaluation in hot deformation, J.of Materials Processing Technology, v 53, n 3-4, Sep 1995,. pp. 552-566.

Rosenblatt, F., The Perceptron: A probabilistic model for information storage and organization in the brain, Psychological Review, 65, 1958, pp. 386-408.

Rosenblatt, F. (1962).Principles of Neurodynamics. Spartan, Chicago.

Rumelhart, D. E. and McClelland, J. L. (1986). *Parallel Distributed Processing*, MIT Press, Cambridge, MA.

Rumelhart, D. E., Hinton, G. E., and Williams, R. J. , Learning representations by back-propagating errors, Nature, 323, 1986, pp. 533-536.

Schauwinhold, D. and Schluter, W., Development of steel types and steel grades as a fraction of structure, Stahl and Eisen, March 19, 1984, pp. 39- 46.

Schmitter, E. D., Automatic grain size determination and classification of iron carbides with neural nets, Steel Research, v 66, n 10, Oct 1995, pp 449-453.

Schnelle, K. D. and Mah, R. S. H., A Real-time Expert System for Quality Control, IEEE Expert, Oct, 1992, pp. 36-42.

Shannon, C.E. (1950). Programming a computer for playing chess. Philosophical Magzine, 41 (4), pp.256-275.

Shepherd, G. M., and Koch, C., Introduction to synaptic circuits, The Synaptic Organization of the Brain, 1990, Oxford University Press, pp. 3-31.

Shynk, J.J., Performance surfaces of a single- layer perceptron, IEEE Transactions on Neural Networks, 1, 1990, pp. 268-274.

Staib, W.E., Bliss, N. G. ,Metallurgical Plant and Technology International, v 18, n 2, Apr 1995, 3pp.

Sun, X.G., Chang, C., Liu, X.H., Wang, G.D., Yu, B., Han, F., Identification of load distribution for finishing mill by artificial neural network, Kang T'ieh/Iron and Steel (Peking), v 30, n 11, Nov 1995,. pp 26-29.

Svolou, A. and Hudak, J. (1987). An Expert System Prototype For Fault Diagnosis in a Rolling Mill, Conf. Record IEEE IAS Annual Meeting 1987, pp. 1081-1086.

Turing, A.M.,Strachey, C., Bates, M. A., and Bowden, B.V. (1953). Digital computers applied to games.*Faster Than Thought*, pp. 286-310.

Werbros, P.J., Beyond regression: New tools for prediction and analysis in the behavioral sciences, Ph.D. Thesis, Havard University, 1974.

Widrow, B. and Hoff, M. E. (1960). Adaptive switching circuits. In *1960 IRE WESCON Convention Record*, pp. 96-104, NY.

Widrow, B. (1962). Generalization and information storage in networks of adaline "neurons". *Self-Organizing Systems 1962*, pp. 435-461, Chicago, IL.

Widrow, B., and Stearns, S. D., Adaptive Signal Processing, Prentice-Hall 1985.

Wieland, A. and Leighton, R., Geometric analysis of neural netowrk capabilities, 1st IEEE Int. Conf on Neural Networks, V 3, 1987, pp.385-392.

Winograd, S. and Cowan, J. D. (1963). *Reliable Computation in the Presence of Noise*. MIT Press, Cambridge, Massachusetts.

Yokoi, T., Kuribayashi, T., Tanimoto, Y., Kamata, T. and Mori, K., Applied technology of Artificial Intelligence and its history for Sumitomo Metal's applications, The Sumitomo Search, No. 50, July, 1992, pp. 3-10.

Appendix A

Introduction to NeuralWorks Professional II/PLUS

NeuralWorks Professional II/ PLUS (Fig. A.1) from NeuralWare Inc. is a complete and comprehensive multi-paradigm prototyping and development system. It can be used to design, build, train, test and deploy neural networks to solve complex, real-world problems.

It offers over two dozen well known network types, including the back-propagation network that we used. The networks are displayed through network diagrams or Hinton diagrams. The network performance can be monitored with the instruments offered by the NeuralWorks.

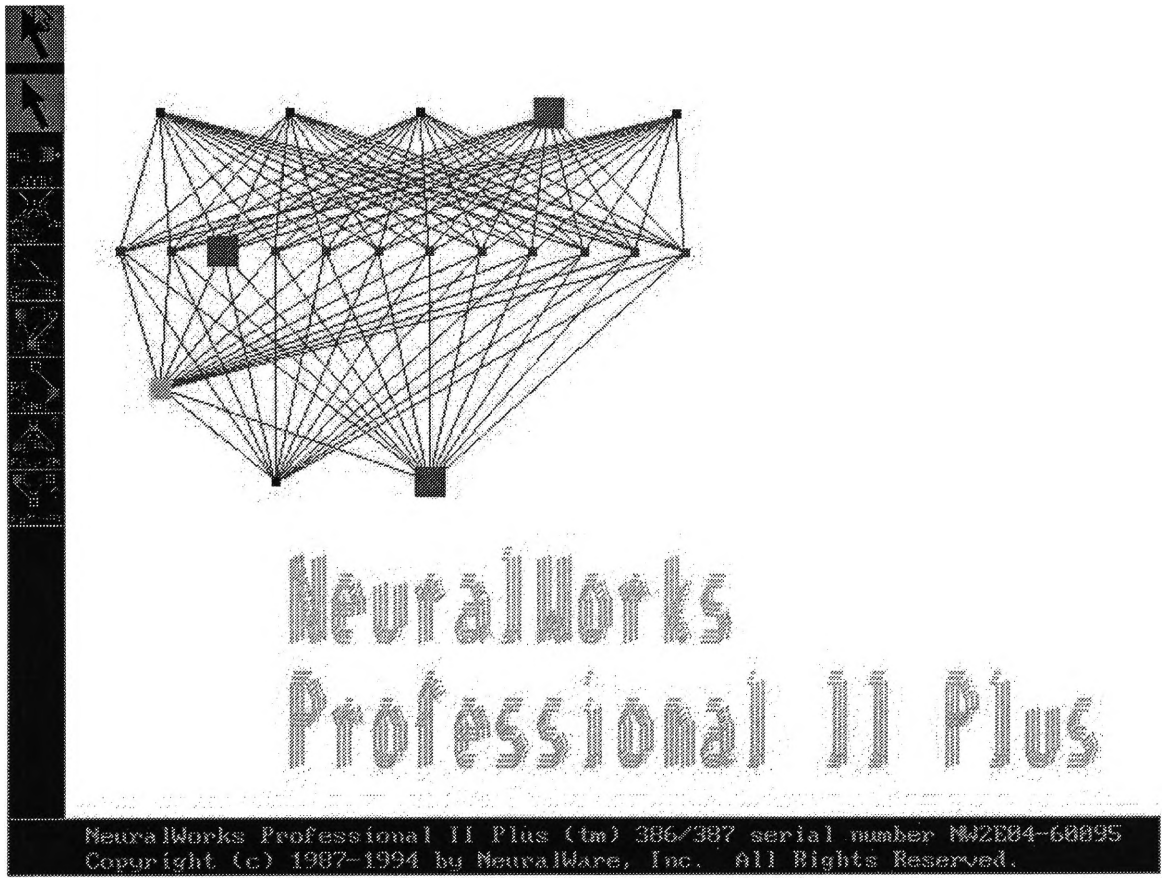


Fig. A.1 Front page of NeuralWork Professional II/PLUS

Fig. A.2 Main menu

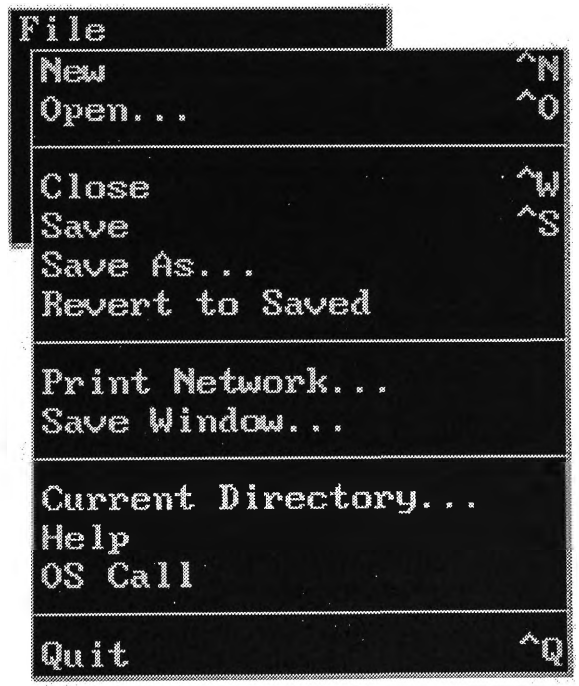


Fig. A.3 File menu

1. The menus

The main menu is shown in Fig. A.2.

The file menu, shown in Fig. A.3, gives control over the major file operations such as save, open, print and quit. In addition, it has commands that let you create a printable graphics file of the current contents of window, print a table listing the components and neuro-dynamics of the current network, and access printer control functions.

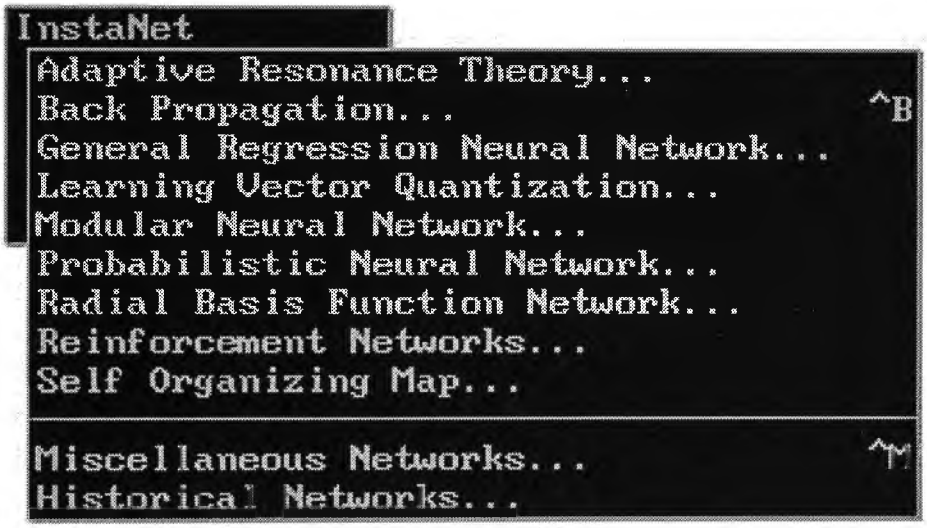


Fig. A.4 The InstNet menu

The InstaNet menu in Fig. A.4, allows selection from any of the major network types to create any of the major paradigms. Each type has its own dialogue box with options and features specific to its type. To build a back-propagation network, the number of PEs[1], the learning rule, the transfer function, the learn and test data files, and the epoch size

[1] PE, Processing Element, is the neural network node called in NeuralWorks Professional II/ PLUS.

need to be specified, as well as some other options. The dialogue box is shown in Fig. A.5.

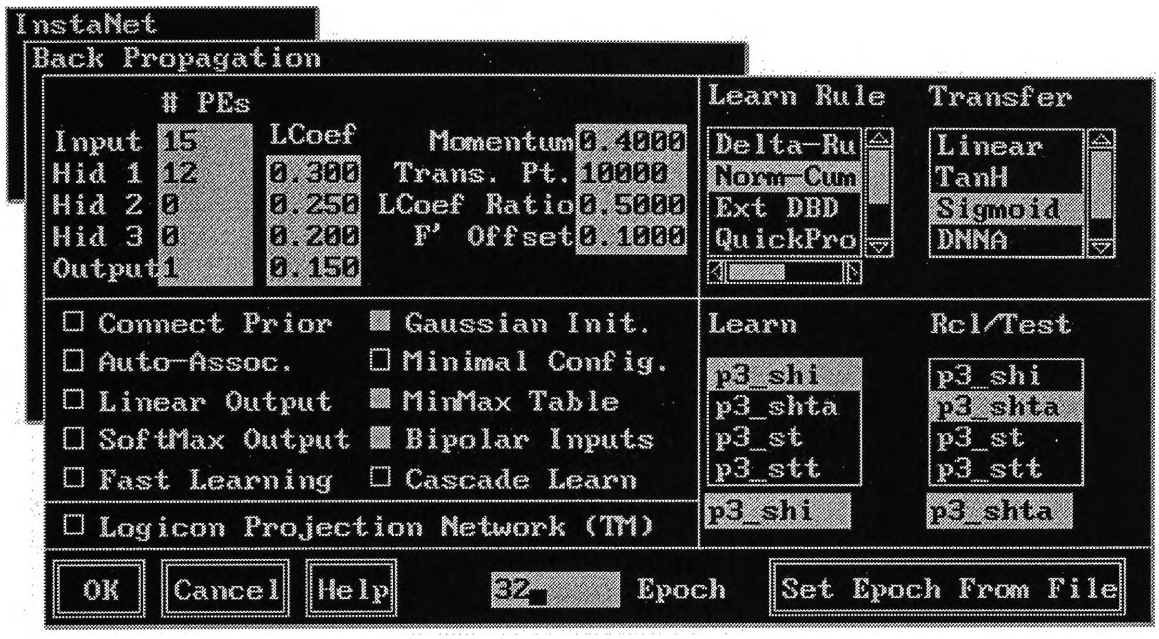


Fig. A.5 Back-propagation install dialogue box

After pressing OK in the Back-propagation dialogue box, an instrument selection dialogue box will pop up. It provides a list of instruments that may be particularly useful with back-propagation networks.

I/O Menu:

The Parameters dialogue box specifies the source and destination of network data, as well as how to preprocess the data so that the input and output values are appropriate for the neural network being used. The I/O Parameters dialogue box, as in Fig. A.6, allows choices on the order in which training data is presented, whether or not binary files are to be used or created, and other options.

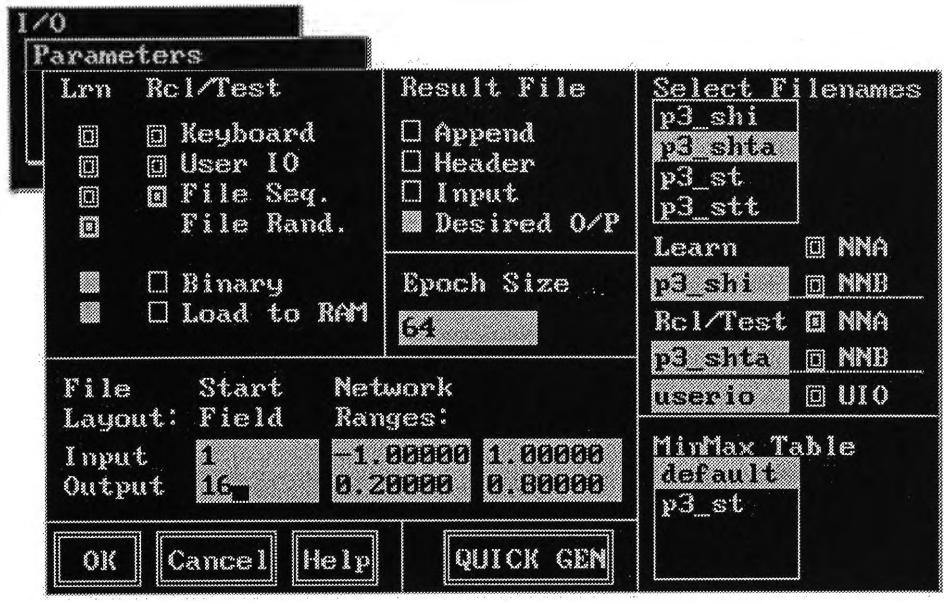


Fig. A.6 I/O menu, Parameters dialogue box

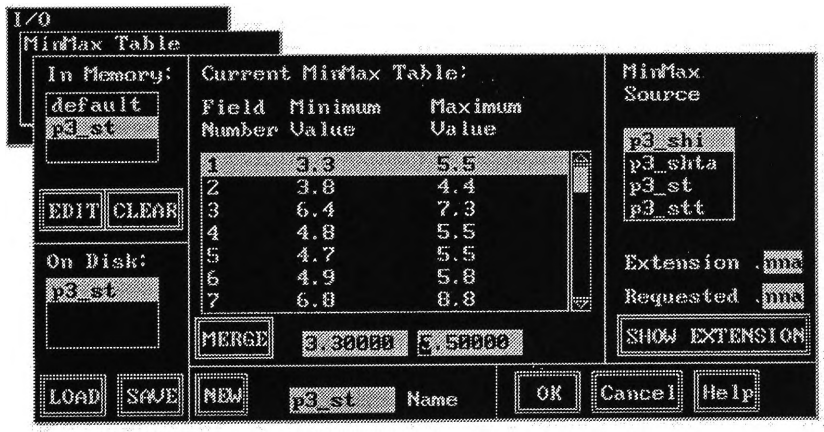


Fig. A.7 I/O menu, MinMax Table dialogue box

The MinMax Table dialogue box allows the creation of MinMax tables; i.e., tables of the minimum and maximum values for each variable in one or more data files. The information in these tables, as listed in Fig. A.7, is used to preprocess raw data. The selected Network MinMax Table specified in the I/O/ Parameters dialogue box is used in conjunction with the Input and Output data Start Columns and the Input and Output Network ranges to create scales and offsets which map real world features or measurements to values acceptable to a network.

The Run Menu allows the network to work. Learn tells NeuralWorks how to let network proceed with learning. Test mode tells how well the network has learned. Initialise Network randomises incoming variable weights to each layer. Either a Uniform or Gaussian distribution is used depending on the selected noise function for a given layer.

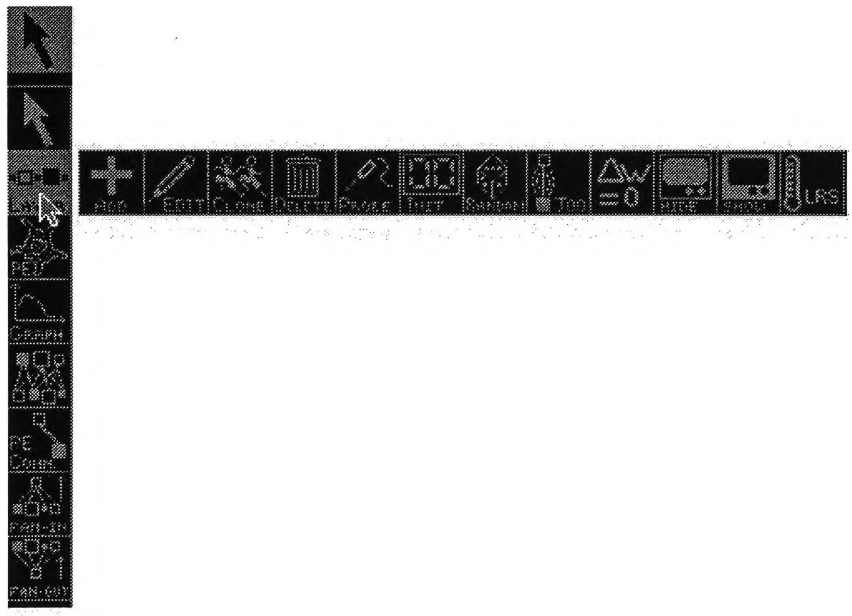


Fig. A.8 Layer Palette

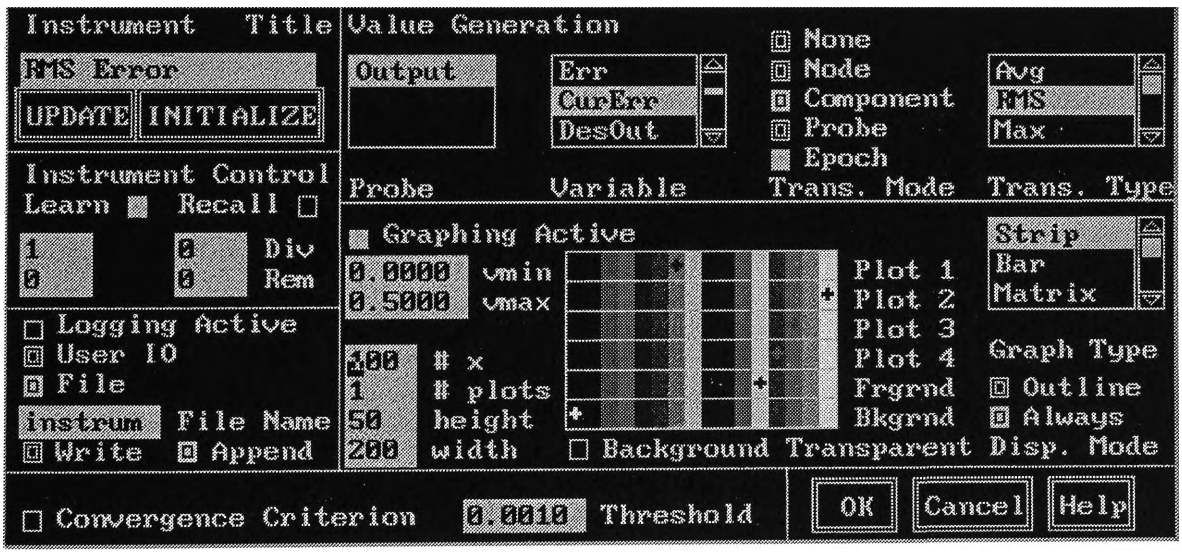
2. Palettes

The layer tool palette, shown in fig. A.8, gives the power of controlling the layer in the network, adding, editing, deleting and copying layers. The Learn/ Recall Schedule Tool is very important. This tool allows editing, loading, saving or creating a learning and recall schedule for a particular layer. This was described in Chapter 5. Other palettes provides similar editing power to PE and PE connections.

3. Instruments

One of the keys to training a neural network is in being able to monitor the weights, processing element values and network performance as the network evolves. With this data, one can make informed decisions about the proper network topology, mathematical functions, training times, learning parameters, etc... NeuralWorks has a powerful, flexible and easy-to-use diagnostic information from a network to the screen.

The RMS Error Instrument measures the Root Mean Square (RMS)[2] Error for all the PEs in the output layer. This RMS error is a valuable and common measure of the performance of a network during training. To examine the RMS Error instrument, double click on the RMS Error Graph or select Edit tool from Graph palette. The edit dialog box for the instrument is shown in Fig. A.9.



[2] The root mean square error adds up the squares of the errors for each PE in the output layer, divides by the number of PEs in the output layer to obtain an average, and then takes the square root of that average. Hence the name root mean square.

Fig. A.9 Instrument dialogue box

There are other instruments used for back-propagation network. They are the Weight Histogram instrument, the Classification Rate instrument and the Confusion Matrix

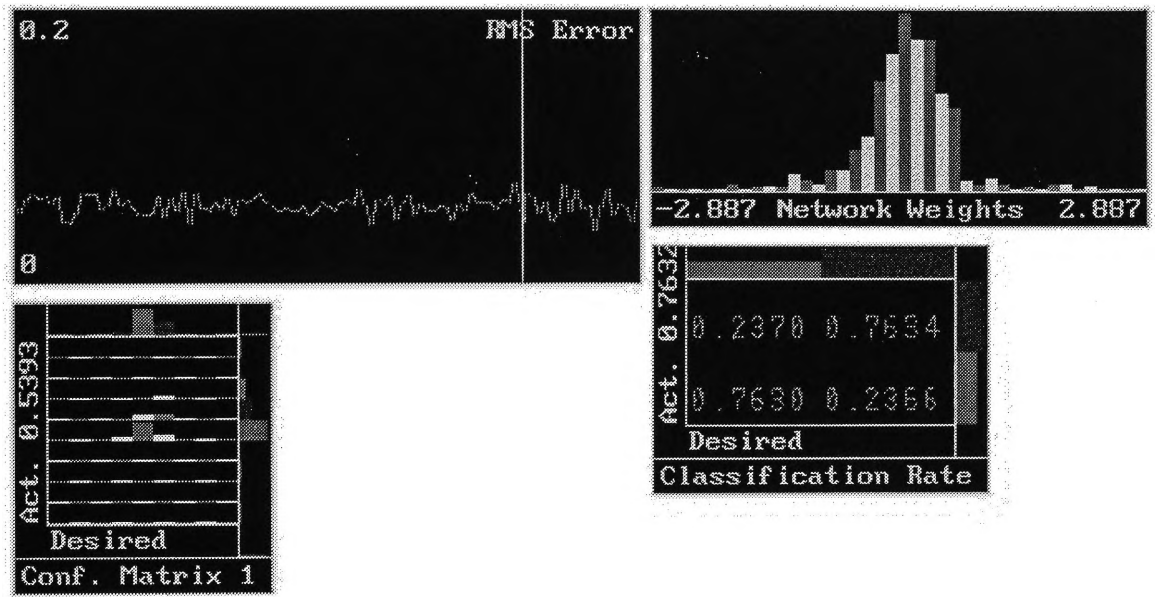


Fig. A.10 Instruments for Back-propagation network

instrument. All of them are very useful in visualising the training performance of the network. Fig. A.10 gives an example of all the instruments.

Appendix B

Software Configuration of SCXI Logging System

The configuration of the board is accomplished through software. This configuration software , WDAQCONF, is a software running under Microsoft Windows system. Fig B.1 to Fig. B.3 show some of the configuration windows.

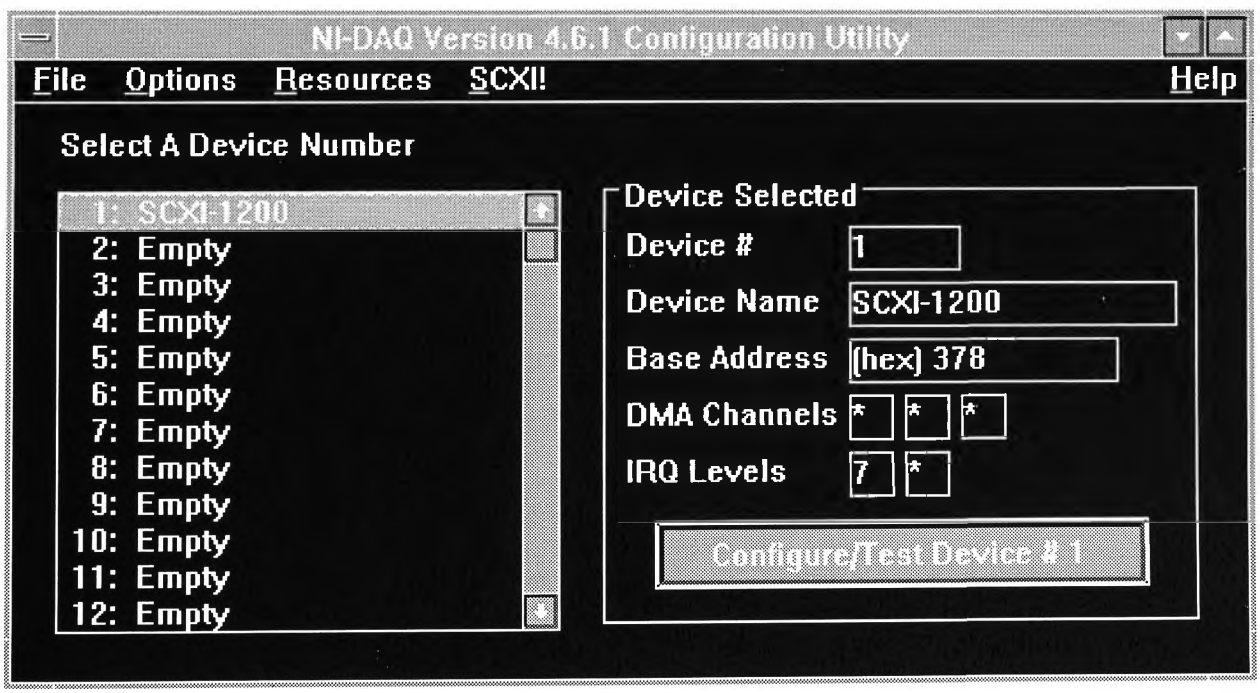


Fig. B.1 The software configuration of SCXI system

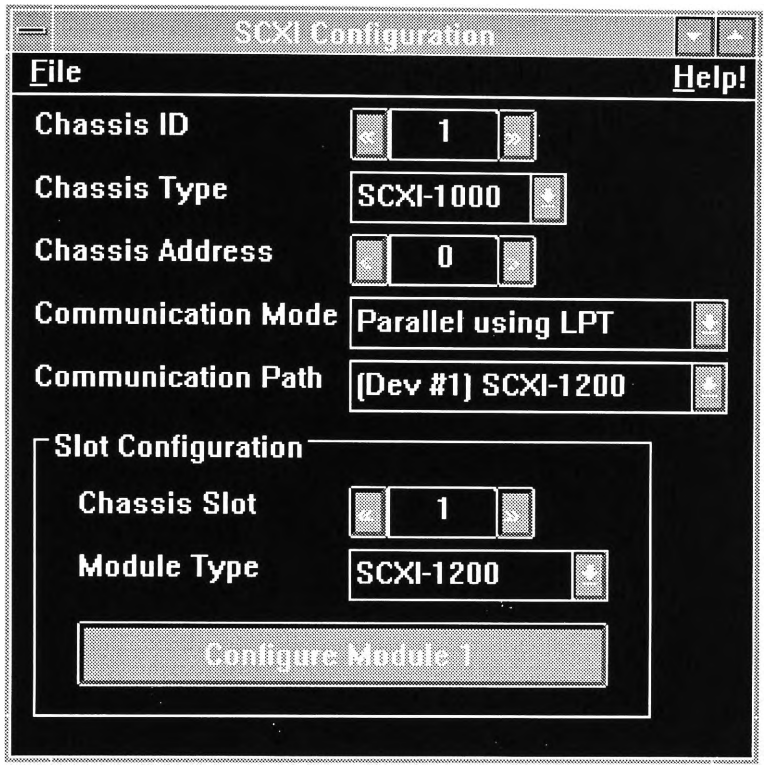


Fig.B.2 Configuration of Chassis SCXI-1000

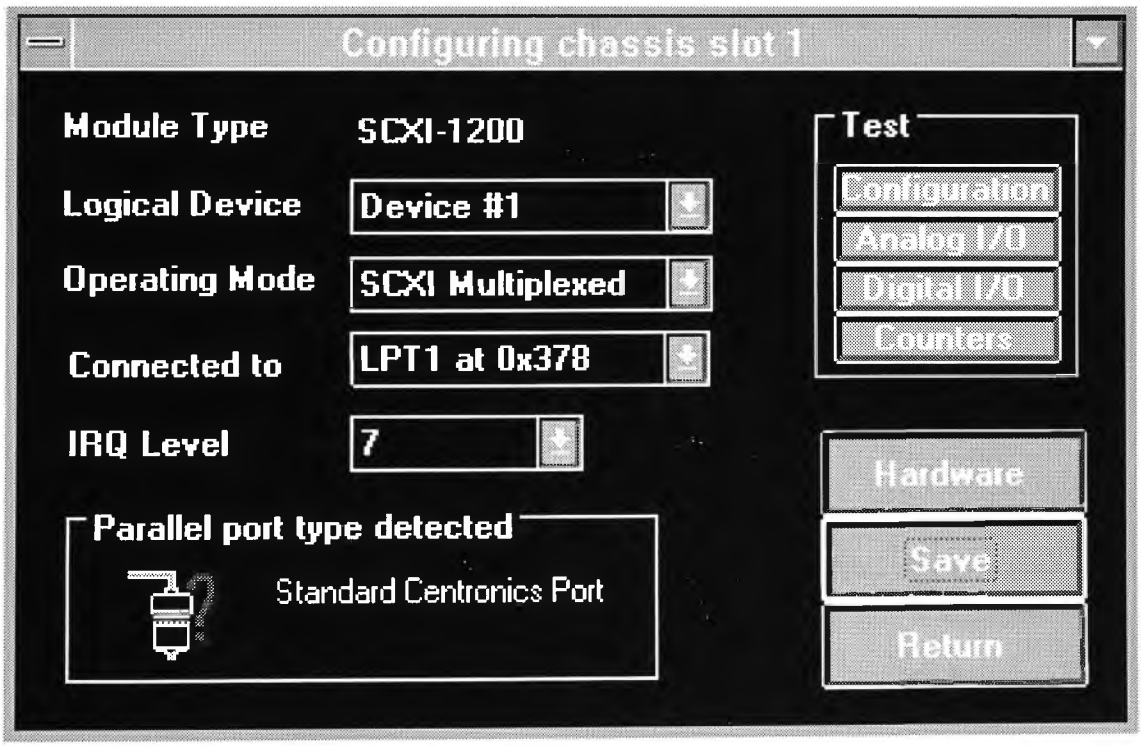


Fig. B.3 Configuration of SCXI-1200

Appendix C

FIR Digital Filter Design and Application

The programs here are for the digital filter used in Chapter 4. The transfer function for the filter

$$H(z) = B(z) = b_0 + b_1 z^{-1} + \ldots + b_L z^{-L}$$

or in the time domain,

$$y_k = \sum_{n=0}^{L} b_n x_{k-n}$$

This equation describes an FIR filter. Some subroutines for the programs are revised from spalib.c provided by Stearns and David[3].

[3] Stearns, S.D. and Dvaid, R. A., Signal Processing Algorithms in Fortran and C, 1993, P T R Prentice-Hall, Inc.

1. Filter design

The file, design.cpp, is the program for design the parameters of b_k. Once all the b_k is calculated, the FIR filter is ready for filtering the noise. The following program is written in Turbo C++ 2.0.

```
/*Design.cpp: the program to design the FIR filter parameters b_k */
#include<stdio.h>
#include<math.h>

#define FALSE   0
#define TRUE    1
#define BIG             1e10
#define SMALL   1e-10
#define ORDER5  1e-5
#define ORDER4  1e-4
#define ABS(x) ((x) >= 0 ? (x) : -(x))
#define MIN(a,b) ((a) <= (b) ? (a) : (b))
#define MAX(a,b) ((a) >= (b) ? (a) : (b))

typedef struct {
                float r, i;
                } complex;

double spwndo(long type, long n, long k);
double complex_abs(complex  z);
complex spgain(float *b, float *a, long lb, long la, float freq);
void spfirl(long l, float fcn, long wndo, float *b, long *error);
void pfile2(float x0, float dx, float *y, long number_points, long number_plots, char *name);
void complex_div(complex *quotient, complex *numerator, complex *denominator);
void complex_exp(complex *r, complex *z);

/* design low pass filter,
b_n = h(n) = h_d(n)w(n),        0 ≤ n ≤ L
The h_d(n) is the ideal impulse-response sequence, and w(n) is any window function.

fc      = the cutoff frequency of the filter,
fcn     = the normalised cutoff frequency, between 0.0 and 0.5 Hz-s.
t       = sampling time, the inverse of sampling frequency
wndo    = data window used.
                1- rectangular          2- tapered rectangular
                3- triangular           4- Hanning
                5- Hamming              6- Blackman
number  = the size of filter, i.e., the maximal number of k for b_k.

spfirl()is the function to calculate the b parameters;
spgain() is used to compute the magnitude response of the resulting digital filters.
pfile2() save all the response from spgain to a file. */
int main(void)
```

```
{
 float b[100], a[1], amp [501];
 long l,wndo,*error,la;
 float fc, fcn,t;
 float freq;
 complex z;

//input cutoff frequency, unit: Hz.
printf("input cutoff frequency:");
scanf("%f",&fc);

//input sampling time, t=1/sampling frequeny, unit: second
printf("sampling time:");
scanf("%f",&t);

// input the size of filter
int number;
printf("filter size:");
scanf("%d",&number);

 a[0]=0;
 fcn=fc*t;
 l=number;
 wndo=5;  // window is set to Hamming, it can be changed to other window as need.

//call the function that calculate the filter parameter.
 spfirl(l,fcn,wndo,b,error);

/* if any error, return error
        no error, continue to calculate the magnitude response of the resulting digital filter */

 if((*error)!=0)
        {
        printf("error returned, error=%d",*error);
        }

  else
  {
   for (int i=0;i<500;i++)
        {
        la=1;
        freq=i*.5/500;
        z=spgain(b,a,l,la,freq);
        amp[i]=complex_abs(z);
        }

 }

//      Continu on save the magnitude into file, named as "filter.dat".

 float x0, dx;
 long np;
 x0=0., dx=1.;
 np=501;
 l=1;
 pfile2(x0,dx,amp,np,l,"filter");

// open a new file "b.dat", save the filter parameters into b.dat.
 FILE * fp1=fopen("b.dat","w+");
 printf("i  b(n)\n");

 for (int i=0;i<number;i++)
 {
 printf("%d        %f\n",i,b[i]);
  fprintf(fp1,"%f\n",b[i]);
  }
```

```
 fclose(fp1);
 return 0;
 }

/* define the subroutine to computecomplex absolute value*/
double complex_abs(complex z)
{
  return( sqrt(z.r * z.r + z.i * z.i) );
}

/* subroutine to put the magnitude values into file

Input    x0          = Starting abscissa value, x[0]
                dx          = x increment
         y[0:number_points,   number_plots]=  Ordinate  values  y[0]  through  y[number_points)]  for
number_plots curves
         number_points    = number of (x,y) points on each curve
         number_plots     = number of separate curve.
         name             = name of file.

*/
void pfile2(float x0, float dx, float *y, long number_points, long number_plots, char *name)
{
  FILE *fp;
  char filename[11];
  long i, j;

sprintf(filename,"%s.DAT",name);

fp = fopen(filename,"w+");

fprintf(fp, "%6ld%6ld \n", number_points, number_plots);
  for ( i = 0 ; i < number_points ; i++ )
         {
         fprintf(fp,"%15.7E", (x0 + (i * dx)));
                for ( j = number_plots-1 ; j >= 0 ; j-- )
                {
                fprintf(fp,"%15.7E", y[i + (j * number_points)]);
                }
         fprintf(fp,"\n");
         }
fclose(fp);
}

/*This function computes the complex gain of any causal liner.You must specify array b, a, complex gain. The
linear system is assumed to have the transfer function.

         B(0)+B(1)*Z**(-1)+B(2)*Z**(-2)+...+B(LB)*Z**(-LB)
H(Z)=-------------------------------------------------------------------
         .0+A(1)*Z**(-1)+A(2)*Z**(-2)+...+A(LA)*Z**(-LA)

for any system , use la=1, and a[1]=0, donot use la=0, freq=frequency in Hz-s, i.e. sampling frequency=1.0*/

complex spgain(float *b, float *a, long lb, long la, float freq)
{
   /* Local variables */
   long i;
   complex asum, bsum, ret_val, tmp_complex1, tmp_complex2, z1;
   float tmp_float;

   tmp_complex1.r = 0.0;
   tmp_complex1.i = (float) (-2.0 * M_PI * freq);
   complex_exp(&z1, &tmp_complex1);

   bsum.r = 0.0;
   bsum.i = 0.0;
```

```
  if (lb > 0)
  {
          for (i = lb ; i >= 1 ; --i)
          {
             tmp_float = (bsum.r + b[i]) * z1.r - bsum.i * z1.i;
             bsum.i = (bsum.r + b[i]) * z1.i + bsum.i * z1.r;
             bsum.r = tmp_float;
          }
  }
  asum.r = 0.0;
  asum.i = 0.0;
  for (i = la - 1 ; i >= 0 ; --i)
  {
          tmp_float = (asum.r + a[i]) * z1.r - asum.i * z1.i;
          asum.i = (asum.r + a[i]) * z1.i + asum.i * z1.r;
          asum.r = tmp_float;
  }
  tmp_complex1.r = 1.0 + asum.r;
  tmp_complex1.i = asum.i;
  if (complex_abs(tmp_complex1) < SMALL)
  {
          ret_val.r = BIG;
          ret_val.i = 0.0;
  }
  else
  {
          tmp_complex1.r = b[0] + bsum.r;
          tmp_complex1.i = bsum.i;
          tmp_complex2.r = 1.0 + asum.r;
          tmp_complex2.i = asum.i;

          complex_div(&ret_val, &tmp_complex1, &tmp_complex2);
  }

   return(ret_val);
} /*end of spgain */

/* This subroutine is for the computing of complex exponential function, the result is return by array r.*/
void complex_exp(complex *r, complex *z)
{
          double expx;

          expx = exp((double) z->r);

          r->r = (float) expx * cos((double) z->i);
          r->i = (float) expx * sin((double) z->i);
}

/*This subroutine is for the computing of complex quotient quotient= numerator/denominator*/
void complex_div(complex *quotient, complex *numerator, complex *denominator)
{
    double ratio, divisor;
   double real, imaginary;

   if( (real = denominator->r) < 0.0)
          real = -real;

   if( (imaginary = denominator->i) < 0.0)
          imaginary = -imaginary;

   if( real <= imaginary )
   {
          if(imaginary == 0)
          {
             ratio = (double) (1.0 / denominator->r);
```

```
            quotient->r = (float) (numerator->r * ratio);
            quotient->i = (float) (numerator->i * ratio);
        }
        else
        {
            ratio = (double) (denominator->r / denominator->i);
            divisor = (double) denominator->i * (1.0 + ratio * ratio);

            quotient->r = (float) ((numerator->r * ratio+ numerator->i) / divisor);
            quotient->i = (float) ((numerator->i * ratio- numerator->r) / divisor);
        }
    }
    else
    {
        if(real == 0)
        {
            ratio = (double) (1.0 / denominator->i);
            quotient->r = (float) (numerator->i * ratio);
            quotient->i = (float) (-numerator->r * ratio);
        }
        else
        {
            ratio = (double) (denominator->i / denominator->r);

            divisor = (double) denominator->r * (1.0 + ratio * ratio);

            quotient->r = (float) ((numerator->r + numerator->i * ratio) / divisor);
            quotient->i = (float) ((numerator->i - numerator->r * ratio) / divisor);
        }
    }
}

/*FIR lowpass filter design windowed fourier series,
l       = filter size such that f(z) = b0) + b(1)*z**(-1) +...+ b(l*z**(-l)
fcn     =nomalized cutoff  frequency in Hz-s
wndo    = data window used.
                1- rectangular          2- tapered rectangular
                3- triangular           4- Hanning
                5- Hamming              6- Blackman
b[0:k]  = filter coefficients returned
error   =0      no errors detected
        =1      invalid filter length (l<=0)
        =2      invalid window type wndo
        =3      invalid cutoff  fcn<=0 OR >=0.5 */

void spfirl(long l, float fcn, long wndo, float *b, long *error)
{
    long i, lim, tmp_int;
    float wcn, dly;
    if (l <= 0)
    {
        *error = 1;
        return;
    }
    if (wndo < 1 || wndo > 6)
    {
        *error = 2;
        return;
    }
    if (fcn <= 0.0 || fcn >= 0.5)
    {
        *error = 3;
        return;
    }
    for (i = 0 ; i <= l ; ++i)
    {
```

```
            b[i] = 0.0;
    }
    wcn = (float) (2.0 * M_PI * fcn);
    dly = (float) l / 2.0;
    lim = l / 2;
    if (dly == (float) (l / 2))
    {
            --lim;
            b[l / 2] = (float) (wcn / M_PI);
    }
    for (i = 0 ; i <= lim ; ++i)
    {
            tmp_int = 1+ l;
            b[i] = (float) (sin((double) (wcn *  ((float) i - dly)))/ (M_PI * ((float) i - dly))
                                    * spwndo(wndo, tmp_int, i));
            b[l - i] = b[i];
    }
    *error = 0;
    return;
} /* end of spfirl */

/* This function generates a single sample of data window
n= size (total number samples) of window
k=sample number withoin window, frm 0 through n-1.
(If k is outside this range, window is set to */

double spwndo(long type, long n, long k)
{
    /* Local variables */
    long l;
    double ret_val;
    if ((type < 1 || type > 6) || (k < 0 || k >= n))
    {
            ret_val = 0.0;
            return(ret_val);
    }
    ret_val = 1.0;
    switch (type)
    {
            case 1:
                    break;
            case 2:
                    l = (n - 2) / 10;
                    if (k <= l)
                    {
                    ret_val = 0.5 * (1.0 - cos((double) k * M_PI
                                                    / ((double) l + 1.0)));
                    }
                    if (k > (n - l - 2))
                    {
                    ret_val = 0.5 * (1.0 - cos((double) (n - k - 1) * M_PI
                                                    / ((double) l + 1.0)));
                    }
                    break;
            case 3:
                    ret_val = 1.0 - ABS(1.0 - (double) (k * 2) / ((double) n - 1.0));
                    break;
            case 4:
                    ret_val = 0.5 * (1.0 - cos((double) (k * 2) * M_PI/ ((double) n - 1.0)));
                    break;
            case 5:
                    ret_val = 0.54 - 0.46 * cos((double) (k * 2) * M_PI/ ((double) n - 1.0));
                    break;
            case 6:
                    ret_val = 0.42 - 0.5 * cos((double) (k * 2) * M_PI/ ((double) n - 1.0))
                            + 0.08 * cos((double) (k * 4) * M_PI/ ((double) n - 1.0));
```

```
                    break;
            }
    return(ret_val);
} /* end of spwndo */
```

2. Application of digital FIR filter.

Once the filter parameters are design, there is a file, *b.dat*, generated. With this file, we can use it to do the filtering. The following programs, *fil.cpp* and *filter.cpp*, is written Turbo c++, running in Dos system. The two files make up a project *fil.prj*. The *fil.cpp* file contains the main() function. Fil.cpp is as follows.

```
/* Fil.cpp: main of the filter program*/
#include <stdio.h>
#include <stdlib.h>

int filt(float *data, int n);
int main(void)
{
 char name1[14], name2[14],name3[14];
 FILE *fp1, *fp2,*fp3;
 int e_o_f;
 float x1[1000];
 float x2[1000],x3[1000],x4[1000],x5[1000];
 float x6[1000],x7[1000],x8[1000],x9[1000],x10[1000],x11[1000];
 float x12[1000],x0[1000];

/* open data file to be filtered*/
 printf(" File to be filtered:");
 scanf("%s", name1);
  if ((fp1 = fopen(name1, "r"))== NULL)
 {
   fprintf(stderr, "Cannot open input file.\n");
   exit (0);
 }

 int i=0;
 for( e_o_f=0; e_o_f!=1;) //i= counter, i++ place after scanf to prevent error
 {
            fscanf(fp1,"%f",x0+i);
            char c=getc(fp1);
            if(c==EOF)
            {
            e_o_f=1;
            break;
            }

fscanf(fp1,"%f\t%f\t%f\t%f\t%f\t%f\t%f\t%f\t%f\t%f\t%f\t%f\t%f",\
     x2+i,x3+i,x4+i,x5+i,x6+i,x7+i,x8+i,x9+i,x10+i,x11+i,x12+i,x1+i);

// minus the offset from sensor (PLCs) signal
  x0[i]=x0[i]-4.0;
  x1[i]=x1[i]-4.0;
  x2[i]=x2[i]-4.0;
  x3[i]=x3[i]-4.0;
  x4[i]=x4[i]-4.0;
  x5[i]=x5[i]-4.0;
  x6[i]=x6[i]-4.0;
  x7[i]=x7[i]-4.0;
  x8[i]=x8[i]-4.0;
  x9[i]=x9[i]-4.0;
  x10[i]=x10[i]-4.0;
```

```
 x11[i]=x11[i]-4.0;
 x12[i]=x12[i]-4.0;
 i+=1;

  c=getc(fp1);
  if(c==EOF) e_o_f=1;
 }
int n=i;
 fclose(fp1);

//input the name of file to save the filtered data
 printf("save filtered data to:");
 scanf("%s",name2);

   if ((fp3 = fopen(name2, "w+")) == NULL)
   { fprintf(stderr, "Cannot open saving filter file\n");
    exit(0);
   }
/*filter the data channel by channel*/
 filt(x0, n);
 filt(x1, n);
 filt(x2,n);
 filt(x3, n);
 filt(x4, n);
 filt(x5,n);
 filt(x6, n);
 filt(x7, n);
 filt(x8,n);
 filt(x9, n);
 filt(x10, n);
 filt(x11,n);
 filt(x12,n);
```

/* For a lowpass filter, the magnitude for signals whose frequency is below the cutoff frequency fc, should be 1. Sometimes the filter design have a scale on magnitude, which means, the magnitude of frequency below cutoff frequency is not 1 but some fraction. An amplitude coefficient need to be multiplied to make the signal remain the same magnitude for cutoff frequency.
The value for s here is set to the inverse of the magnitude of cutoff frequency that we used in a filter. It should vary from one filter to another.*/

```
float s=1./.26;
  for(i=0;i<n;i++){
   x0[i]=x0[i]*s;
   x1[i]=x1[i]*s;
   x2[i]=x2[i]*s;
   x3[i]=x3[i]*s;
   x4[i]=x4[i]*s;
   x5[i]=x5[i]*s;
   x6[i]=x6[i]*s;
   x7[i]=x7[i]*s;
   x8[i]=x8[i]*s;
   x9[i]=x9[i]*s;
   x10[i]=x10[i]*s;
   x11[i]=x11[i]*s;
   x12[i]=x12[i]*s;
                }

// save the filtered data to file
  for(i=32;i< n-32;i++)
 {
  fprintf(fp3,"%f\t%f\t%f\t%f\t%f\t%f\t%f\t%f\t%f\t%f\t%f\t%f\t%f\t%f\t%f\n",\
     x0[i],x2[i],x3[i],x4[i],x5[i],x6[i],\
     x7[i],x8[i],x9[i],x10[i],x11[i],x12[i],x1[i]);
   }

 fclose(fp3);
```

```
 return 0;
}
```

filter.cpp is as follows.

```
/* filter.cpp: the subroutines for filtering*/

#include<stdio.h>
#include<stdlib.h>

#define FALSE   0
#define TRUE    1
#define BIG     1e10
#define SMALL   1e-10
#define ORDER5  1e-5
#define ORDER4  1e-4
#define ABS(x) ((x) >= 0 ? (x) : -(x))
#define MIN(a,b) ((a) <= (b) ? (a) : (b))
#define MAX(a,b) ((a) >= (b) ? (a) : (b))


void spfilt(float *b, float *a, int lb,int la, float *x,int n, float *px, float *py, int *error);

//start main program.
int filt(float *data, int num)
{
  float x[100],b[100],a[1],px[100],py[1];
  int *error, f_n;
  char name[14];
  FILE * fp1, *fp2, *fp3;

   f_n=32;

// open the filter parameters file
    if ((fp2 = fopen("b.dat", "r")) == NULL)
    { fprintf(stderr, "Cannot open file <b.dat>.\n");
      exit(0);
    }

// read all the parameters from b.dat.
  for(int i=0; i<f_n;i++)
  { fscanf(fp2,"%f", b+i);
  }
   fclose(fp2);

 a[0]=0.;
 int la,lb,n;

 la=0;
 lb=f_n-1;
 n=32;


/* initialize all the variables px[], py[0]=0;*/
  for(i=0; i<f_n;i++)
  {
   px[i]=0.;
   }
   py[0]=0.;

 int  d_l=num/n;
 for(int j=0;j< d_l;j++)
 {
 /*put data into x for filtering;*/
    for(i=0; i<n;i++)
   {
```

```
 x[i]=data[n*j+i];
 }

 spfilt(b,a,lb,la,x,n,px,py,error);
 if((*error)!=0)
 {
  printf("error returned, error=%d",*error);
 }
/* put filtered data back to x */
 for (i=0;i<n;i++)
 {
  data[n*j+i]=x[i];
 }

 }

 return 0;
}

void spfilt(float *b, float *a,int lb, int la, float *x, int n, float *px, float *py, int *error)
{
   long k, l;

   for (k = 0 ; k < n ; ++k)
   {
            px[0] = x[k];
            x[k] = 0.0;
            for (l = 0 ; l <= lb ; ++l)
            {
               x[k] += b[l] * px[l];
            }
            for (l = 0 ; l < la ; ++l)
            {
               x[k] -= a[l] * py[l];
            }
            if (ABS(x[k]) > BIG)
            {
               *error = 1;
               return;
            }
            for (l = lb ; l >= 1 ; --l)
            {
               px[l] = px[l - 1];
            }
            for (l = la - 1 ; l >= 1 ; --l)
            {
               py[l] = py[l - 1];
            }
            py[0] = x[k];
   }
   *error = 0;
   return;
} /* spfilt */
```

Appendix D

The Signal Processing Program

The following program is for the signal processing described in chapter 4. In section 4.2.2, only the processing of forward pass is discussed, as the processing of reversing pass is similar to that of forward passes. Here the program includes processing for both forward and reversing passes.

Since the program is too long, it is made into a project, *proc.prj*, which contains several files. The main function is in file *pmain.cpp*. Other three are *sub1.cpp, sub2.cpp* and *sub3.cpp*. All the programs are written in C++, the compiler is Turbo C++ 2.0.

The program is going to process the data file logged from PLCs signals, which has 13 signals. There may be lots of interim files, which will be deleted in the end. The output files would still be data files with 13 signals.The channel signals are listed below in Table D-1.

Table D-1 List of signals in data file

On	Pannel	Test	Variables
0	RMW	0	WG1 Deviaton (Entry)
1	RMW	1	WG2 Deviation (Exit)
2	RMW	2	RE Gap Deviation
3	RMW	3	RE Force
4	RMC	0	RE-Speed
5	RMC	1	RE Current
6	RMC	2	RR Bottom Speed
7	RMC	3	RR Exit Temperature
8	RMC	4	RR-RE Tension
9	RMC	5	RR D/S Force
10	RMC	6	RR O/S Force
11	RMC	7	RR top Current
12	RMC	8	RR Bottom Current

// The main file, PMAIN.CPP

```
#include <stdio.h>
#include <graphics.h>
#include <math.h>
#include <stdlib.h>
#include <string.h>

//define of prototype of subroutines, as required by Turbo C++.
int div_chan_file(char *);
int  cut_chan_3( int *, int *);
int  cut_chan_9( int *, int *);
int cut_chan_7 (int *, int *,int *);
int cut_chan_8(int *,int *);
int cut_reverse(int *, int *);
int cut_chan_1(int *, int *, int *);
int cut_chan_0(int *, int *,int *,int *);
int bb_p2(char *,int , float );
int bb_p4(char *,int , float );
int in_p3(char *,int ,int , float );
int in_p5(char *,int ,int , float );
int in_p7(char *,int ,int , float );
int clean();
int smt(int , float *, float *);

int main()
{
 int  edger_start[]={0,0,0,0,0,0,0,0}, edger_end[]={0,0,0,0,0,0,0,0};
 int  RR_start[]={0,0,0,0,0,0,0,0}, RR_end[]={0,0,0,0,0,0,0,0};
 int  temp_start[]={0,0,0,0,0,0,0,0}, temp_end[]={0,0,0,0,0,0,0,0};
char in_f_name[32];

/* input the signal file being logged */
 printf("input data file name:");
scanf("%s",in_f_name);

/*divide into the single file into 13 files, each channel one file*/
  div_chan_file(in_f_name);

 //divide channel 3 into 1,3,5,7 pass; also channel 2,4,5 of 1357 pass
 cut_chan_3(edger_start, edger_end);
```

```
//cut channel 9 10 11 12 6
 cut_chan_9( RR_start, RR_end);

// cut channel 7
cut_chan_7 (temp_start,temp_end, RR_start);

//cut channel 8
cut_chan_8(RR_start, edger_end);

// cut channel 4 5 reversing 246 passes
cut_reverse(RR_start,RR_end);

// cut width channels 0
cut_chan_0(RR_start,RR_end,edger_start,edger_end);

//cut channel 1;
cut_chan_1(RR_end,temp_start,temp_end);

 //make  training file for NN
  int t2=RR_end[1]-RR_start[1];
  if((bb_p2(in_f_name,t2,1.5))!=0)
  {
   printf("Fail to make pass2 bar-bar file");
  }
int   t1=edger_end[2]-edger_start[2];
  t2=RR_end[2]-RR_start[2];
  if((in_p3(in_f_name,t1,t2,1.5))!=0)
  {
   printf("Fail to make pass3 in bar file");
  }
   t2=RR_end[3]-RR_start[3];
  if((bb_p4(in_f_name,t2,1.5))!=0)
  {
   printf("Fail to make pass4 bar-bar file");
  }
  t1=edger_end[4]-edger_start[4];
  t2=RR_end[4]-RR_start[4];
 if((in_p5(in_f_name,t1,t2,1.5))!=0)
  {
   printf("Fail to make pass 5 in bar file");
  }

  t1=edger_end[6]-edger_start[6];
  t2=RR_end[6]-RR_start[6];
 if((in_p7(in_f_name,t1,t2,1.5))!=0)
  {
   printf("Fail to make pass 7 in bar file");
  }
 clean();
 return 0;
 } // finish the main function, below are all subroutines.
// make the 13 channel sample file into 13 seperate files
 int  div_chan_file(char * name)
{
  FILE *fp1;
  FILE *fd1, *fd2, *fd3, *fd4, *fd5, *fd6, *fd7, *fd8, *fd9, *fd10,\
      *fd11, *fd12, *fd0;
  float cha1, cha2, cha3, cha4, cha5 ,cha6, cha7, cha8, cha9, cha10, cha11,
            cha12, cha0;
   int i,j,read_pointer=0;
  char ch ;
  char fn[38];

//the input name.dat is the filename, fn.
  strcpy(fn,name);
  strncat(fn,".dat", 5);
if ((fp1 = fopen(fn, "r"))
```

```
  == NULL)
 {
  fprintf(stderr, "Cannot open input file.\n");
  exit  (0);
  }
//create divided channel data file, ch0.dat to ch12.dat.
 if ((fd0 = fopen("ch0.dat", "w+")) == NULL)
  { fprintf(stderr, "Cannot open file <ch0.dat>.\n");
   exit(0);
  }
 if ((fd1 = fopen("ch1.dat", "w+")) == NULL)
   {  fprintf(stderr, "Cannot open file <ch1.dat>.\n");
    exit(0);
  }
  if ((fd2 = fopen("ch2.dat", "w+")) == NULL)
    { fprintf(stderr, "Cannot open file <ch2.dat>.\n");
     exit(0);
    }
  if ((fd3 = fopen("ch3.dat", "w+")) == NULL)
    { fprintf(stderr, "Cannot open file <ch3.dat>.\n");
     exit(0);
    }
   if ((fd4 = fopen("ch4.dat", "w+")) == NULL)
     { fprintf(stderr, "Cannot open file <ch4.dat>.\n");
      exit(0);
     }
   if ((fd5 = fopen("ch5.dat", "w+")) == NULL)
    { fprintf(stderr, "Cannot open file <ch5.dat>.\n");
     exit(0);
    }
   if ((fd6 = fopen("ch6.dat", "w+")) == NULL)
    { fprintf(stderr, "Cannot open file <ch6.dat>.\n");
     exit(0);
   }
   if ((fd7 = fopen("ch7.dat", "w+")) == NULL)
     { fprintf(stderr, "Cannot open file <ch7.dat>.\n");
    exit(0);
    }
   if ((fd8 = fopen("ch8.dat", "w+")) == NULL)
     { fprintf(stderr, "Cannot open file <ch8.dat>.\n");
     exit(0);
    }
   if ((fd9 = fopen("ch9.dat", "w+")) == NULL)
     { fprintf(stderr, "Cannot open file <ch9.dat>.\n");
     exit(0);
     }
    if ((fd10 = fopen("ch10.dat", "w+")) == NULL)
     {  fprintf(stderr, "Cannot open file <ch10.dat>.\n");
           exit(0);
     }
    if ((fd11 = fopen("ch11.dat", "w+")) == NULL)
      { fprintf(stderr, "Cannot open file <ch11.dat>.\n");
            exit(0);
     }
    if ((fd12 = fopen("ch12.dat", "w+")) == NULL)
      { fprintf(stderr, "Cannot open file <ch12.dat>.\n");
           exit(0);
     }
  // read the original file, and write into channel data files
   for(;read_pointer!=1;)
  {
   fscanf(fp1, "%f", &cha0);
   fscanf(fp1, "%f", &cha1);
   fscanf(fp1, "%f", &cha2);
   fscanf(fp1, "%f", &cha3);
   fscanf(fp1, "%f", &cha4);
   fscanf(fp1, "%f", &cha5);
```

```
  fscanf(fp1, "%f", &cha6);
  fscanf(fp1, "%f", &cha7);
  fscanf(fp1, "%f", &cha8);
  fscanf(fp1, "%f", &cha9);
  fscanf(fp1, "%f", &cha10);
  fscanf(fp1, "%f", &cha11);
  fscanf(fp1, "%f", &cha12);

  fprintf(fd0,"%.3f\n", cha0);
  fprintf(fd1,"%.3f\n", cha1);
  fprintf(fd2,"%.3f\n", cha2);
  fprintf(fd3,"%.3f\n", cha3);
  fprintf(fd4,"%.3f\n", cha4);
  fprintf(fd5,"%.3f\n", cha5);
  fprintf(fd6,"%.3f\n", cha6);
  fprintf(fd7,"%.3f\n", cha7);
  fprintf(fd8,"%.3f\n", cha8);
  fprintf(fd9,"%.3f\n", cha9);
  fprintf(fd10,"%.3f\n", cha10);
  fprintf(fd11,"%.3f\n", cha11);
  fprintf(fd12,"%.3f\n", cha12);

  ch=fgetc(fp1);
  if(ch==EOF)
  {
   read_pointer=1;
  printf("End of file\n");
  }
 }

//close all files.
 if(( fclose(fd0))==0)
 {printf("close successfully file 'ch0'\n");
 }
  fclose(fd1);
  fclose(fd2);
  fclose(fd3);
  fclose(fd4);
  fclose(fd5);
  fclose(fd6);
  fclose(fd7);
  fclose(fd8);
  fclose(fd9);
  fclose(fd10);
  fclose(fd11);
  fclose(fd12);
  fclose(fp1);

  return 0;
}
//end of file pmain.cpp

/*file sub1.cppstart*/
#include <stdio.h>
#include <graphics.h>
#include <math.h>
#include <stdlib.h>

int smt(int , float *, float *);

/* This subroutine identifies the forward passes of channel 3, RE rolling force. Since channel2, channel 4,
channel 5 is synchrinized with channel 3, they are also identified by forward passes.*/
int  cut_chan_3(int *p_s, int *p_e)
{
int i;
FILE *fd3, *ch3;
float d[5]={4.0,4.0,4.0,4.0,4.0},s1=0.;
```

```
float dd[5]={0.,0.,0.,0.,0.};
int start,stop=0,sample;
float d2_1,d2_2,d4_1,d4_2,d5_1,d5_2;

  FILE *fd2, *fd4, *fd5;
  FILE *ch2;
  FILE *ch4;
  FILE *ch5;

  if ((fd3 = fopen("ch3.dat", "r")) == NULL)
  { fprintf(stderr, "Cannot open file <ch3.dat>.\n");
    exit(0);
  }
  if ((fd2 = fopen("ch2.dat", "r")) == NULL)
  { fprintf(stderr, "Cannot open file <ch2.dat>.\n");
    exit(0);
  }
  if ((fd4 = fopen("ch4.dat", "r")) == NULL)
  { fprintf(stderr, "Cannot open file <ch4.dat>.\n");
    exit(0);
  }
  if ((fd5 = fopen("ch5.dat", "r")) == NULL)
  { fprintf(stderr, "Cannot open file <ch5.dat>.\n");
    exit(0);
  }
  if ((ch2 = fopen("ch2_p1.dat", "w+")) == NULL)
  { fprintf(stderr, "Cannot open file <ch2_p1.dat>.\n");
    exit(0);
  }
  if ((ch3 = fopen("ch3_p1.dat", "w+")) == NULL)
  { fprintf(stderr, "Cannot open file <ch3_p1.dat>.\n");
    exit(0);
  }
  if ((ch4 = fopen("ch4_p1.dat", "w+")) == NULL)
  { fprintf(stderr, "Cannot open file <ch4_p1.dat>.\n");
    exit(0);
  }
  if ((ch5 = fopen("ch5_p1.dat", "w+")) == NULL)
  { fprintf(stderr, "Cannot open file <ch5_p1.dat>.\n");
    exit(0);
  }

/* judge the start and end of channel 3 signal.
start=0 before the real signal start, it is set to 1 when it begins to write the first data into file and increase after
that.
sample is set to 0 at each reading, it will set to 1 , when writing the channel file.
stop determine when to stop writing this pass.
The algorithm for other passes is similar to that of pass 1.
*/

//cut pass1
 for(start=0,i=0;stop!=1;i=i+2)
 {
  sample=0;
   d[4]=d[2],d[2]=d[0],d[3]=d[1];
  fscanf(fd3,"%f\n%f\n", d,d+1);
   smt(5, d, dd); //call the subroutine to calculate smoothed values
   s1=( dd[0]+dd[1]+dd[2]+dd[3]+dd[4])/5.;
  fscanf(fd2,"%f\n%f\n", &d2_1,&d2_2);
  fscanf(fd4,"%f\n%f\n", &d4_1,&d4_2);
  fscanf(fd5,"%f\n%f\n", &d5_1,&d5_2);

if(dd[0]>5.0 && s1>4.9)
  {
   fprintf(ch3,"%f\n%f\n", d[0],d[1]);
   fprintf(ch2,"%f\n%f\n", d2_1,d2_2);
   fprintf(ch4,"%f\n%f\n", d4_1,d4_2);
```

```
  fprintf(ch5,"%f\n%f\n", d5_1,d5_2);

  start++;
  sample=1;
 }
  if(start==1)p_s[0]=i;
  if(start!=0 && sample==0)stop=1,p_e[0]=i; //determine when to stop.
 }

//close all pass files.
  fclose(ch3);
  fclose(ch2);
  fclose(ch4);
  fclose(ch5);
   printf("edger_start_p1=%d\tp1_e1=%d\n", p_s[0],p_e[0]);

//cut pass3
d[2]=4.0, d[3]=4.0, d[4]=4.0;
stop=0;

   if ((ch2 = fopen("ch2_p3.dat", "w+")) == NULL)
    { fprintf(stderr, "Cannot open file <ch2_p3.dat>.\n");
     exit(0);
    }

   if ((ch3 = fopen("ch3_p3.dat", "w+")) == NULL)
    { fprintf(stderr, "Cannot open file <ch3_p3.dat>.\n");
     exit(0);
    }

   if ((ch4 = fopen("ch4_p3.dat", "w+")) == NULL)
    { fprintf(stderr, "Cannot open file <ch4_p3.dat>.\n");
     exit(0);
    }
   if ((ch5 = fopen("ch5_p3.dat", "w+")) == NULL)
    { fprintf(stderr, "Cannot open file <ch5_p3.dat>.\n");
     exit(0);
    }
   for(start=0,i=0;i<200;i=i+2)
 {
  fscanf(fd3,"%f\n%f\n", d,d+1);
  fscanf(fd2,"%f\n%f\n", &d2_1,&d2_2);
  fscanf(fd4,"%f\n%f\n", &d4_1,&d4_2);
  fscanf(fd5,"%f\n%f\n", &d5_1,&d5_2);
  }
  for(start=0,i=0;stop!=1;i=i+2)
 {
  sample=0;
   d[4]=d[2],d[2]=d[0],d[3]=d[1];
  fscanf(fd3,"%f\n%f\n", d,d+1);
  smt(5, d, dd);
   s1=( dd[0]+dd[1]+dd[2]+dd[3]+dd[4])/5.;
  fscanf(fd2,"%f\n%f\n", &d2_1,&d2_2);
  fscanf(fd4,"%f\n%f\n", &d4_1,&d4_2);
  fscanf(fd5,"%f\n%f\n", &d5_1,&d5_2);

  if(dd[0]>5.0 && s1>4.9)
  {
   fprintf(ch3,"%f\n%f\n", d[0],d[1]);
   fprintf(ch2,"%f\n%f\n", d2_1,d2_2);
   fprintf(ch4,"%f\n%f\n", d4_1,d4_2);
   fprintf(ch5,"%f\n%f\n", d5_1,d5_2);  sample=0;

   start++;
   sample=1;
  }
  if(start==1)
```

```
  {p_s[2]=p_e[0]+i+200;
  }
   if(start!=0 &&sample==0)stop=1,p_e[2]=p_e[0]+i+200;
 }
  fclose(ch3);
  fclose(ch2);
  fclose(ch4);
  fclose(ch5);
   printf("edger_start_p3=%d\tp3_e1=%d\n", p_s[2],p_e[2]);

 //cut pass5
d[2]=4.,d[3]=4.,d[4]=4.;
stop=0;

   if ((ch2 = fopen("ch2_p5.dat", "w+")) == NULL)
    { fprintf(stderr, "Cannot open file <ch2_p5.dat>.\n");
     exit(0);
    }

   if ((ch3 = fopen("ch3_p5.dat", "w+")) == NULL)
    { fprintf(stderr, "Cannot open file <ch3_p5.dat>.\n");
     exit(0);
    }

   if ((ch4 = fopen("ch4_p5.dat", "w+")) == NULL)
    { fprintf(stderr, "Cannot open file <ch4_p5.dat>.\n");
     exit(0);
    }

   if ((ch5 = fopen("ch5_p5.dat", "w+")) == NULL)
    { fprintf(stderr, "Cannot open file <ch5_p5.dat>.\n");
     exit(0);
    }

   for(start=0,i=0;i<500;i=i+2)
 {
  fscanf(fd3,"%f\n%f\n", d,d+1);
  fscanf(fd2,"%f\n%f\n", &d2_1,&d2_2);
  fscanf(fd4,"%f\n%f\n", &d4_1,&d4_2);
  fscanf(fd5,"%f\n%f\n", &d5_1,&d5_2);
  }

  for(start=0,i=0;stop!=1;i=i+2)
 {
    sample=0;
   d[4]=d[2],d[2]=d[0],d[3]=d[1];
  fscanf(fd3,"%f\n%f\n", d,d+1);
  smt(5, d, dd);
   s1=( dd[0]+dd[1]+dd[2]+dd[3]+dd[4])/5.;

  fscanf(fd2,"%f\n%f\n", &d2_1,&d2_2);
  fscanf(fd4,"%f\n%f\n", &d4_1,&d4_2);
  fscanf(fd5,"%f\n%f\n", &d5_1,&d5_2);

  if(dd[0]>4.7 && s1>4.5)
  {
   fprintf(ch3,"%f\n%f\n", d[0],d[1]);
   fprintf(ch2,"%f\n%f\n", d2_1,d2_2);
   fprintf(ch4,"%f\n%f\n", d4_1,d4_2);
   fprintf(ch5,"%f\n%f\n", d5_1,d5_2);

   start++;
   sample=1;
  }

   if(start==1)p_s[4]=p_e[2]+i+500;
  if(start!=0&&sample==0)
```

```
 {
 p_e[4]= p_e[2]+i+500;
  if((p_e[4]-p_s[4]) < 250) {
/* in case of wrongly judge, redo it*/
    start=0;
    rewind(ch3);
    rewind(ch2);
    rewind(ch4);
    rewind(ch5);
   }
  else stop=1;
 }
}
 fclose(ch3);
 fclose(ch2);
 fclose(ch4);
 fclose(ch5);
 printf("edger start_p5=%d\tp5_e1=%d\n", p_s[4],p_e[4]);

//cut pass7
d[2]=4.,d[3]=4.,d[4]=4.;
stop=0;

  if ((ch2 = fopen("ch2_p7.dat", "w+")) == NULL)
   { fprintf(stderr, "Cannot open file <ch2_p7.dat>.\n");
    exit(0);
   }
  if ((ch3 = fopen("ch3_p7.dat", "w+")) == NULL)
   { fprintf(stderr, "Cannot open file <ch3_p7.dat>.\n");
    exit(0);
   }
  if ((ch4 = fopen("ch4_p7.dat", "w+")) == NULL)
   { fprintf(stderr, "Cannot open file <ch4_p7.dat>.\n");
    exit(0);
   }
  if ((ch5 = fopen("ch5_p7.dat", "w+")) == NULL)
   { fprintf(stderr, "Cannot open file <ch5_p7.dat>.\n");
    exit(0);
   }
  for(start=0,i=0;i<1000;i=i+2)
{
 fscanf(fd3,"%f\n%f\n", d,d+1);
 fscanf(fd2,"%f\n%f\n", &d2_1,&d2_2);
 fscanf(fd4,"%f\n%f\n", &d4_1,&d4_2);
 fscanf(fd5,"%f\n%f\n", &d5_1,&d5_2);
 }
 for(start=0,i=0;stop!=1;i=i+2)
{
 sample=0;
  d[4]=d[2],d[2]=d[0],d[3]=d[1];
 fscanf(fd3,"%f\n%f\n", d,d+1);
 smt(5, d, dd);
  s1=( dd[0]+dd[1]+dd[2]+dd[3]+dd[4])/5.;
 fscanf(fd2,"%f\n%f\n", &d2_1,&d2_2);
 fscanf(fd4,"%f\n%f\n", &d4_1,&d4_2);
 fscanf(fd5,"%f\n%f\n", &d5_1,&d5_2);

 if(dd[0]>4.10 &&dd[1]>4.10 &&dd[2]>4.10 &&dd[5]>4.10 && s1>4.10)
 {
 fprintf(ch3,"%f\n%f\n", d[0],d[1]);
 fprintf(ch2,"%f\n%f\n", d2_1,d2_2);
 fprintf(ch4,"%f\n%f\n", d4_1,d4_2);
 fprintf(ch5,"%f\n%f\n", d5_1,d5_2);
 start++;
 sample=1;
 }
  if(start==1)p_s[6]=p_e[4]+i+1000;
```

```
 if(start!=0 &&sample==0)
 {
  p_e[6]= p_e[4]+i+1000;
  if((p_e[6]-p_s[6]) < 500) {
   start=0;
   rewind(ch3);
   rewind(ch2);
   rewind(ch4);
   rewind(ch5);
  }
  else stop=1;
 }
 }
fclose(fd3);
fclose(fd2);
fclose(fd4);
fclose(fd5);
fclose(ch3);
fclose(ch2);
fclose(ch4);
fclose(ch5);
 printf("edger_start_p7=%d\tp7_e1=%d\n", p_s[6],p_e[6]);
 return 0;
}//end of subrountine cut_chan_3().

/*cut _chan_9, this subroutine indentidy channel 9, 10, 11, 12 and 6 for all passes. The algorithm is similar to
that of cut_chan_3.*/
int  cut_chan_9( int *ps, int *pe)
{
int i;
float d[5]={4.0,4.0,4.0,4.0,4.0},s1=0.;
float dd[5]={0.,0.,0.,0.,0.};
int start,stop=0,sample;
float  d6_1,d6_2,d10_1,d10_2,d11_1,d11_2,d12_1,d12_2;

  FILE *fd6, *fd9, *fd10,*fd11, *fd12 ;
  FILE  *ch6;
  FILE  *ch9;
  FILE  *ch10;
  FILE  *ch11;
  FILE  *ch12;

  if ((fd6 = fopen("ch6.dat", "r")) == NULL)
  { fprintf(stderr, "Cannot open file <ch6.dat>.\n");
    exit(0);
  }
  if ((fd9 = fopen("ch9.dat", "r")) == NULL)
  { fprintf(stderr, "Cannot open file <ch9.dat>.\n");
    exit(0);
  }
  if ((fd10 = fopen("ch10.dat", "r")) == NULL)
  { fprintf(stderr, "Cannot open file <ch10.dat>.\n");
    exit(0);
  }
  if ((fd11 = fopen("ch11.dat", "r")) == NULL)
  { fprintf(stderr, "Cannot open file <ch11.dat>.\n");
    exit(0);
  }
  if ((fd12 = fopen("ch12.dat", "r")) == NULL)
  { fprintf(stderr, "Cannot open file <ch12.dat>.\n");
    exit(0);
  }
  if ((ch6 = fopen("ch6_p1.dat", "w+")) == NULL)
  { fprintf(stderr, "Cannot open file <ch6_p1.dat>.\n");
    exit(0);
  }
  if ((ch9 = fopen("ch9_p1.dat", "w+")) == NULL)
```

```
    { fprintf(stderr, "Cannot open file <ch9_p1.dat>.\n");
      exit(0);
    }
    if ((ch10 = fopen("ch10_p1.dat", "w+")) == NULL)
    { fprintf(stderr, "Cannot open file <ch10_p1.dat>.\n");
      exit(0);
    }
    if ((ch11 = fopen("ch11_p1.dat", "w+")) == NULL)
    { fprintf(stderr, "Cannot open file <ch11_p1.dat>.\n");
      exit(0);
    }
   if ((ch12 = fopen("ch12_p1.dat", "w+")) == NULL)
    { fprintf(stderr, "Cannot open file <ch12_p1.dat>.\n");
      exit(0);
    }
 //cut pass1
  for(start=0,i=0;stop!=1;i=i+2)
 {
  sample=0;
   d[4]=d[2],d[2]=d[0],d[3]=d[1];
  fscanf(fd9,"%f\n%f\n", d,d+1);
  smt(5, d, dd);
   s1=( dd[0]+dd[1]+dd[2]+dd[3]+dd[4])/5.;
  fscanf(fd6,"%f\n%f\n", &d6_1,&d6_2);
  fscanf(fd10,"%f\n%f\n", &d10_1,&d10_2);
  fscanf(fd11,"%f\n%f\n", &d11_1,&d11_2);
  fscanf(fd12,"%f\n%f\n", &d12_1,&d12_2);
  if(dd[0]>5.2 && s1>5.2)
  {
   fprintf(ch9,"%f\n%f\n", d[0],d[1]);
   fprintf(ch6,"%f\n%f\n", d6_1,d6_2);
   fprintf(ch10,"%f\n%f\n", d10_1,d10_2);
   fprintf(ch11,"%f\n%f\n", d11_1,d11_2);
   fprintf(ch12,"%f\n%f\n", d12_1,d12_2);
   start++;
   sample=1;
  }
   if(start==1)ps[0]=i;
  if(start!=0 && sample==0)stop=1, pe[0]=i;
 }
  fclose(ch6);
  fclose(ch9);
  fclose(ch10);
  fclose(ch11);
  fclose(ch12);
   printf("RR_start_p1=%d\tpe1=%d\n",ps[0],pe[0]);
 //cut pass2
d[2]=4.,d[3]=4.,d[4]=4.;
stop=0;
   if ((ch6 = fopen("ch6_p2.dat", "w+")) == NULL)
    { fprintf(stderr, "Cannot open file <ch6_p2.dat>.\n");
      exit(0);
    }
   if ((ch9 = fopen("ch9_p2.dat", "w+")) == NULL)
    { fprintf(stderr, "Cannot open file <ch9_p2.dat>.\n");
      exit(0);
    }
   if ((ch10 = fopen("ch10_p2.dat", "w+")) == NULL)
    { fprintf(stderr, "Cannot open file <ch10_p2.dat>.\n");
      exit(0);
    }
   if ((ch11 = fopen("ch11_p2.dat", "w+")) == NULL)
    { fprintf(stderr, "Cannot open file <ch11_p2.dat>.\n");
      exit(0);
    }
  if ((ch12 = fopen("ch12_p2.dat", "w+")) == NULL)
    { fprintf(stderr, "Cannot open file <ch12_p2.dat>.\n");
```

```
    exit(0);
   }
   for(start=0,i=0;i<300;i=i+2)
 {
  fscanf(fd9,"%f\n%f\n", d,d+1);
  fscanf(fd6,"%f\n%f\n", &d6_1,&d6_2);
  fscanf(fd10,"%f\n%f\n", &d10_1,&d10_2);
  fscanf(fd11,"%f\n%f\n", &d11_1,&d11_2);
  fscanf(fd12,"%f\n%f\n", &d12_1,&d12_2);
  }
  for(start=0,i=0;stop!=1;i=i+2)
 {
  sample=0;
   d[4]=d[2],d[2]=d[0],d[3]=d[1];
  fscanf(fd9,"%f\n%f\n", d,d+1);
  smt(5, d, dd);
   s1=( dd[0]+dd[1]+dd[2]+dd[3]+dd[4])/5.;
  fscanf(fd6,"%f\n%f\n", &d6_1,&d6_2);
  fscanf(fd10,"%f\n%f\n", &d10_1,&d10_2);
  fscanf(fd11,"%f\n%f\n", &d11_1,&d11_2);
  fscanf(fd12,"%f\n%f\n", &d12_1,&d12_2);

  if(dd[0]>5.2 && s1>5.2)
  {
   fprintf(ch9,"%f\n%f\n", d,d+1);
   fprintf(ch6,"%f\n%f\n", d6_1,d6_2);
   fprintf(ch10,"%f\n%f\n", d10_1,d10_2);
   fprintf(ch11,"%f\n%f\n", d11_1,d11_2);
   fprintf(ch12,"%f\n%f\n", d12_1,d12_2);
   start++;
   sample=1;
  }
   if(start==1)ps[1]=pe[0]+i+300;
  if(start!=0 && sample==0)stop=1, pe[1]=pe[0]+i+300;
 }
  fclose(ch6);
  fclose(ch9);
  fclose(ch10);
  fclose(ch11);
  fclose(ch12);
   printf("RR_start_ps2=%d\tpe2=%d\n",ps[1],pe[1]);

 //cut pass3
d[2]=4.,d[3]=4.,d[4]=4.;
stop=0;
  if ((ch6 = fopen("ch6_p3.dat", "w+")) == NULL)
   { fprintf(stderr, "Cannot open file <ch6_p3.dat>.\n");
    exit(0);
   }
  if ((ch9 = fopen("ch9_p3.dat", "w+")) == NULL)
   { fprintf(stderr, "Cannot open file <ch9_p3.dat>.\n");
    exit(0);
   }
  if ((ch10 = fopen("ch10_p3.dat", "w+")) == NULL)
   { fprintf(stderr, "Cannot open file <ch10_p3.dat>.\n");
    exit(0);
   }
  if ((ch11 = fopen("ch11_p3.dat", "w+")) == NULL)
   { fprintf(stderr, "Cannot open file <ch11_p3.dat>.\n");
    exit(0);
   }
 if ((ch12 = fopen("ch12_p3.dat", "w+")) == NULL)
   { fprintf(stderr, "Cannot open file <ch12_p3.dat>.\n");
    exit(0);
   }
   for(start=0,i=0;i<300;i=i+2)
{
```

```
  fscanf(fd9,"%f\n%f\n", d,d+1);
  fscanf(fd6,"%f\n%f\n", &d6_1,&d6_2);
  fscanf(fd10,"%f\n%f\n", &d10_1,&d10_2);
  fscanf(fd11,"%f\n%f\n", &d11_1,&d11_2);
  fscanf(fd12,"%f\n%f\n", &d12_1,&d12_2);
  }
  for(start=0,i=0;stop!=1;i=i+2)
 {
  sample=0;
   d[4]=d[2],d[2]=d[0],d[3]=d[1];
  fscanf(fd9,"%f\n%f\n", d,d+1);
  smt(5, d, dd);
   s1=( dd[0]+dd[1]+dd[2]+dd[3]+dd[4])/5.;
  fscanf(fd6,"%f\n%f\n", &d6_1,&d6_2);
  fscanf(fd10,"%f\n%f\n", &d10_1,&d10_2);
  fscanf(fd11,"%f\n%f\n", &d11_1,&d11_2);
  fscanf(fd12,"%f\n%f\n", &d12_1,&d12_2);

  if(dd[0]>5.2 && s1>5.2)
  {
   fprintf(ch9,"%f\n%f\n", d[0],d[1]);
   fprintf(ch6,"%f\n%f\n", d6_1,d6_2);
   fprintf(ch10,"%f\n%f\n", d10_1,d10_2);
   fprintf(ch11,"%f\n%f\n", d11_1,d11_2);
   fprintf(ch12,"%f\n%f\n", d12_1,d12_2);
   start++;
   sample=1;
  }

   if(start==1)ps[2]=pe[1]+i+300;
   if(start!=0 && sample==0)stop=1, pe[2]=pe[1]+i+300;
 }

  fclose(ch6);
  fclose(ch9);
  fclose(ch10);
  fclose(ch11);
  fclose(ch12);
    printf("RR_start_ps3=%d\tpe3=%d\n",ps[2],pe[2]);

 //cut pass4
d[2]=4.,d[3]=4.,d[4]=4.;
stop=0;

   if ((ch6 = fopen("ch6_p4.dat", "w+")) == NULL)
    { fprintf(stderr, "Cannot open file <ch6_p4.dat>.\n");
     exit(0);
    }
   if ((ch9 = fopen("ch9_p4.dat", "w+")) == NULL)
    { fprintf(stderr, "Cannot open file <ch9_p4.dat>.\n");
     exit(0);
    }
   if ((ch10 = fopen("ch10_p4.dat", "w+")) == NULL)
    { fprintf(stderr, "Cannot open file <ch10_p4.dat>.\n");
     exit(0);
    }
   if ((ch11 = fopen("ch11_p4.dat", "w+")) == NULL)
    { fprintf(stderr, "Cannot open file <ch11_p4.dat>.\n");
     exit(0);
    }
  if ((ch12 = fopen("ch12_p4.dat", "w+")) == NULL)
    { fprintf(stderr, "Cannot open file <ch12_p4.dat>.\n");
     exit(0);
    }
   for(start=0,i=0;i<300;i=i+2)
 {
  fscanf(fd9,"%f\n%f\n", d,d+1);
```

```
 fscanf(fd6,"%f\n%f\n", &d6_1,&d6_2);
 fscanf(fd10,"%f\n%f\n", &d10_1,&d10_2);
 fscanf(fd11,"%f\n%f\n", &d11_1,&d11_2);
 fscanf(fd12,"%f\n%f\n", &d12_1,&d12_2);
 }
 for(start=0,i=0;stop!=1;i=i+2)
{
 sample=0;
 d[4]=d[2],d[2]=d[0],d[3]=d[1];
 fscanf(fd9,"%f\n%f\n", d,d+1);
 smt(5, d, dd);
 s1=( dd[0]+dd[1]+dd[2]+dd[3]+dd[4])/5.;
 fscanf(fd6,"%f\n%f\n", &d6_1,&d6_2);
 fscanf(fd10,"%f\n%f\n", &d10_1,&d10_2);
 fscanf(fd11,"%f\n%f\n", &d11_1,&d11_2);
 fscanf(fd12,"%f\n%f\n", &d12_1,&d12_2);
 if(dd[0]>5.5 && s1>5.5)
 {
  fprintf(ch9,"%f\n%f\n", d[0],d[1]);
  fprintf(ch6,"%f\n%f\n", d6_1,d6_2);
  fprintf(ch10,"%f\n%f\n", d10_1,d10_2);
  fprintf(ch11,"%f\n%f\n", d11_1,d11_2);
  fprintf(ch12,"%f\n%f\n", d12_1,d12_2);
  start++;
  sample=1;
 }
  if(start==1)ps[3]=pe[2]+i+300;
  if(start!=0 && sample==0)stop=1, pe[3]=pe[2]+i+300;
}
 fclose(ch6);
 fclose(ch9);
 fclose(ch10);
 fclose(ch11);
 fclose(ch12);
   printf("RR_start_ps4=%d\tpe4=%d\n",ps[3],pe[3]);
 //cut pass5
d[2]=4.,d[3]=4.,d[4]=4.;
stop=0;
  if ((ch6 = fopen("ch6_p5.dat", "w+")) == NULL)
   { fprintf(stderr, "Cannot open file <ch6_p5.dat>.\n");
    exit(0);
   }

  if ((ch9 = fopen("ch9_p5.dat", "w+")) == NULL)
   { fprintf(stderr, "Cannot open file <ch9_p5.dat>.\n");
    exit(0);
   }
  if ((ch10 = fopen("ch10_p5.dat", "w+")) == NULL)
   { fprintf(stderr, "Cannot open file <ch10_p5.dat>.\n");
    exit(0);
   }
  if ((ch11 = fopen("ch11_p5.dat", "w+")) == NULL)
   { fprintf(stderr, "Cannot open file <ch11_p5.dat>.\n");
    exit(0);
   }
 if ((ch12 = fopen("ch12_p5.dat", "w+")) == NULL)
   { fprintf(stderr, "Cannot open file <ch12_p5.dat>.\n");
    exit(0);
   }
   for(start=0,i=0;i<300;i=i+2)
{
 fscanf(fd9,"%f\n%f\n", d,d+1);
 fscanf(fd6,"%f\n%f\n", &d6_1,&d6_2);
 fscanf(fd10,"%f\n%f\n", &d10_1,&d10_2);
 fscanf(fd11,"%f\n%f\n", &d11_1,&d11_2);
 fscanf(fd12,"%f\n%f\n", &d12_1,&d12_2);
 }
```

```
 for(start=0,i=0;stop!=1;i=i+2)
 {
  sample=0;
  d[4]=d[2],d[2]=d[0],d[3]=d[1];
  fscanf(fd9,"%f\n%f\n", d,d+1);
  smt(5, d, dd);
  s1=( dd[0]+dd[1]+dd[2]+dd[3]+dd[4])/5.;
  fscanf(fd6,"%f\n%f\n", &d6_1,&d6_2);
  fscanf(fd10,"%f\n%f\n", &d10_1,&d10_2);
  fscanf(fd11,"%f\n%f\n", &d11_1,&d11_2);
  fscanf(fd12,"%f\n%f\n", &d12_1,&d12_2);

  if(dd[0]>5.5 && s1>5.5)
  {
   fprintf(ch9,"%f\n%f\n", d[0],d[1]);
   fprintf(ch6,"%f\n%f\n", d6_1,d6_2);
   fprintf(ch10,"%f\n%f\n", d10_1,d10_2);
   fprintf(ch11,"%f\n%f\n", d11_1,d11_2);
   fprintf(ch12,"%f\n%f\n", d12_1,d12_2);
   start++;
   sample=1;
  }
   if(start==1)ps[4]=pe[3]+i+300;
   if(start!=0 && sample==0)stop=1, pe[4]=pe[3]+i+300;
 }
  fclose(ch6);
  fclose(ch9);
  fclose(ch10);
  fclose(ch11);
  fclose(ch12);
   printf("RR_start_ps5=%d\tpe5=%d\n",ps[4],pe[4]);

 //cut pass6
d[2]=4.,d[3]=4.,d[4]=4.;
stop=0;
   if ((ch6 = fopen("ch6_p6.dat", "w+")) == NULL)
    { fprintf(stderr, "Cannot open file <ch6_p6.dat>.\n");
     exit(0);
    }
   if ((ch9 = fopen("ch9_p6.dat", "w+")) == NULL)
    { fprintf(stderr, "Cannot open file <ch9_p6.dat>.\n");
     exit(0);
    }
   if ((ch10 = fopen("ch10_p6.dat", "w+")) == NULL)
    { fprintf(stderr, "Cannot open file <ch10_p6.dat>.\n");
     exit(0);
    }
   if ((ch11 = fopen("ch11_p6.dat", "w+")) == NULL)
    { fprintf(stderr, "Cannot open file <ch11_p6.dat>.\n");
     exit(0);
    }
  if ((ch12 = fopen("ch12_p6.dat", "w+")) == NULL)
    { fprintf(stderr, "Cannot open file <ch12_p6.dat>.\n");
     exit(0);
    }
    for(start=0,i=0;i<300;i=i+2)
{
 fscanf(fd9,"%f\n%f\n", d,d+1);
 fscanf(fd6,"%f\n%f\n", &d6_1,&d6_2);
 fscanf(fd10,"%f\n%f\n", &d10_1,&d10_2);
 fscanf(fd11,"%f\n%f\n", &d11_1,&d11_2);
 fscanf(fd12,"%f\n%f\n", &d12_1,&d12_2);
}
 for(start=0,i=0;stop!=1;i=i+2)
{
 sample=0;
  d[4]=d[2],d[2]=d[0],d[3]=d[1];
```

```
 fscanf(fd9,"%f\n%f\n", d,d+1);
 smt(5, d, dd);
  s1=( dd[0]+dd[1]+dd[2]+dd[3]+dd[4])/5.;
 fscanf(fd6,"%f\n%f\n", &d6_1,&d6_2);
 fscanf(fd10,"%f\n%f\n", &d10_1,&d10_2);
 fscanf(fd11,"%f\n%f\n", &d11_1,&d11_2);
 fscanf(fd12,"%f\n%f\n", &d12_1,&d12_2);

 if(dd[0]>5.5 && s1>5.5)
 {
  fprintf(ch9,"%f\n%f\n", d[0],d[1]);
  fprintf(ch6,"%f\n%f\n", d6_1,d6_2);
  fprintf(ch10,"%f\n%f\n", d10_1,d10_2);
  fprintf(ch11,"%f\n%f\n", d11_1,d11_2);
  fprintf(ch12,"%f\n%f\n", d12_1,d12_2);
  start++;
  sample=1;
 }
  if(start==1)ps[5]=pe[4]+i+300;
  if(start!=0 && sample==0)stop=1, pe[5]=pe[4]+i+300;
}
 fclose(ch6);
 fclose(ch9);
 fclose(ch10);
 fclose(ch11);
 fclose(ch12);
   printf("RR_start_ps6=%d\tpe6=%d\n",ps[5],pe[5]);

 //cut pass7
d[2]=4.,d[3]=4.,d[4]=4.;
stop=0;
  if ((ch6 = fopen("ch6_p7.dat", "w+")) == NULL)
   { fprintf(stderr, "Cannot open file <ch6_p7.dat>.\n");
    exit(0);
   }
if ((ch9 = fopen("ch9_p7.dat", "w+")) == NULL)
   { fprintf(stderr, "Cannot open file <ch9_p7.dat>.\n");
    exit(0);
   }
  if ((ch10 = fopen("ch10_p7.dat", "w+")) == NULL)
   { fprintf(stderr, "Cannot open file <ch10_p7.dat>.\n");
    exit(0);
   }
  if ((ch11 = fopen("ch11_p7.dat", "w+")) == NULL)
   { fprintf(stderr, "Cannot open file <ch11_p7.dat>.\n");
    exit(0);
   }
 if ((ch12 = fopen("ch12_p7.dat", "w+")) == NULL)
   { fprintf(stderr, "Cannot open file <ch12_p7.dat>.\n");
    exit(0);
   }
   for(start=0,i=0;i<300;i=i+2)
{
 fscanf(fd9,"%f\n%f\n", d,d+1);
 fscanf(fd6,"%f\n%f\n", &d6_1,&d6_2);
 fscanf(fd10,"%f\n%f\n", &d10_1,&d10_2);
 fscanf(fd11,"%f\n%f\n", &d11_1,&d11_2);
 fscanf(fd12,"%f\n%f\n", &d12_1,&d12_2);
}
 for(start=0,i=0;stop!=1;i=i+2)
{
 sample=0;
  d[4]=d[2],d[2]=d[0],d[3]=d[1];
 fscanf(fd9,"%f\n%f\n", d,d+1);
 smt(5, d, dd);
  s1=( dd[0]+dd[1]+dd[2]+dd[3]+dd[4])/5.;
 fscanf(fd6,"%f\n%f\n", &d6_1,&d6_2);
```

```
  fscanf(fd10,"%f\n%f\n", &d10_1,&d10_2);
  fscanf(fd11,"%f\n%f\n", &d11_1,&d11_2);
  fscanf(fd12,"%f\n%f\n", &d12_1,&d12_2);

  if(dd[0]>5.5 && s1>5.5)
  {
   fprintf(ch9,"%f\n%f\n", d[0],d[1]);
   fprintf(ch6,"%f\n%f\n", d6_1,d6_2);
   fprintf(ch10,"%f\n%f\n", d10_1,d10_2);
   fprintf(ch11,"%f\n%f\n", d11_1,d11_2);
   fprintf(ch12,"%f\n%f\n", d12_1,d12_2);
   start++;
   sample=1;
  }
   if(start==1)ps[6]=pe[5]+i+300;
   if(start!=0 && sample==0)stop=1, pe[6]=pe[5]+i+300;
 }
  printf("RR_start_ps7=%d\tpe7=%d\n",ps[6],pe[6]);
   fclose(ch6); fclose(ch9); fclose(ch10); fclose(ch11);
   fclose(ch12); fclose(fd6); fclose(fd9); fclose(fd10);
   fclose(fd11); fclose(fd12);

   return 0;
} // end of the subroutine

/* smt() is the subrountine used to smooth the signal, it is base on the algorithm in section 7.2.2.*/
int smt(int n, float *y, float * yy)
{
 int i;
 if (n<5)
 {for (i=0;i<=n-1;i++) yy[i]=y[i];}
 else {
 yy[0]=69.0*y[0]+4.0*y[1]-6.0*y[2]+4.0*y[3]-y[4];
 yy[0]=yy[0]/70.0;
 yy[1]=2.0*y[0]+27.0*y[1]+12.0*y[2]-8.0*y[3];
 yy[1]=(yy[1]+2.0*y[4])/35.0;
 for (i=2;i<=n-3;i++)
   { yy[i]=-3.0*y[i-2]+12.0*y[i-1]+17.0*y[i];
     yy[i]=(yy[i]+12.0*y[i+1]-3.0*y[i+2])/35.0;
   }
    yy[n-2]=2.0*y[n-5]-8.0*y[n-4]+12.0*y[n-3];
    yy[n-2]=(yy[n-2]+27.0*y[n-2]+2.0*y[n-1])/35.0;
    yy[n-1]=-y[n-5]+4.0*y[n-4]-6.0*y[n-3];
    yy[n-1]=(yy[n-1]+4.0*y[n-2]+69.0*y[n-1])/70.0;
 }
  return 0;
 }//end of smf()
//end of file sub1.cpp


/*file : sub2.cpp start*/

#include <stdio.h>
#include <graphics.h>
#include <math.h>
#include <stdlib.h>

int smt(int , float *, float *);

/*channel 7, exit temperature, is not synchronized with any other signal, so it has to be identified
seperately.The following routine divide this channel into 7 passes.*/
int cut_chan_7 (int *ps, int *pe, int * RR_start)
{
int i;
float d[5]={4.0,4.0,4.0,4.0,4.0},s1=0.;
float dd[5]={0.,0.,0.,0.,0.};
int  start,stop=0,sample;
```

```
   FILE *fd7;
   FILE  *ch7;
    if ((fd7 = fopen("ch7.dat", "r")) == NULL)
    { fprintf(stderr, "Cannot open file <ch7.dat>.\n");
     exit(0);
    }
    if ((ch7 = fopen("ch7_p1.dat", "w+")) == NULL)
     { fprintf(stderr, "Cannot open file <ch7_p1.dat>.\n");
      exit(0);
     }
  //cut pass1
   for(start=0,i=0;stop!=1;i=i+2)
  {
    sample=0;
    d[4]=d[2],d[2]=d[0],d[3]=d[1];
   fscanf(fd7,"%f\n%f\n", d,d+1);
   smt(5, d, dd);
    s1=( dd[0]+dd[1]+dd[2]+dd[3]+dd[4])/5.;

   if(dd[0]>5.5 && s1>5.3)
   {
    fprintf(ch7,"%f\n%f\n", d[0],d[1]);
    start++;
    sample=1;
   }
     if(start==1)ps[0]=i;
   if(start!=0 && sample==0)stop=1, pe[0]=i;
  }
   fclose(ch7);
    printf("temp_start_p1=%d\ttemp_end_p1=%d\n", ps[0], pe[0]);

  //cut pass2
 d[2]=4.,d[3]=4.,d[4]=4.;
 stop=0;
    if ((ch7 = fopen("ch7_p2.dat", "w+")) == NULL)
     { fprintf(stderr, "Cannot open file <ch7_p2.dat>.\n");
      exit(0);
     }
   for(start=0, i=0;i<200;i=i+2)
  {
   fscanf(fd7,"%f\n%f\n", d,d+1);
   }
   for(start=0,i=0;stop!=1;i=i+2)
  {
   sample=0;
    d[4]=d[2],d[2]=d[0],d[3]=d[1];
   fscanf(fd7,"%f\n%f\n", d,d+1);
   smt(5, d, dd);
    s1=( dd[0]+dd[1]+dd[2]+dd[3]+dd[4])/5.;
   if(dd[0]>5.5 && s1>5.3)
   {
    fprintf(ch7,"%f\n%f\n", d[0],d[1]);
    start++;
    sample=1;
   }
     if(start==1)ps[1]=pe[0]+i+200;
   if(start!=0 && sample==0)stop=1, pe[1]=pe[0]+i+200;
  }
   fclose(ch7);
    printf("temp_start_p2=%d\ttemp_end_p2=%d\n", ps[1], pe[1]);
  //cut pass3
 d[2]=4.0,d[3]=4.,d[4]=4.;
 stop=0;
    if ((ch7 = fopen("ch7_p3.dat", "w+")) == NULL)
     { fprintf(stderr, "Cannot open file <ch7_p3.dat>.\n");
      exit(0);
```

```
   }
    for(start=0, i=pe[1];i<RR_start[2];i++)
  {
  fscanf(fd7,"%f\n", d);
  }
  for(start=0,i=0;stop!=1;i=i+2)
  {
  sample=0;
   d[4]=d[2],d[2]=d[0],d[3]=d[1];
  fscanf(fd7,"%f\n%f\n", d,d+1);
  smt(5, d, dd);
   s1=( dd[0]+dd[1]+dd[2]+dd[3]+dd[4])/5.;
  if(dd[0]>5.5 && s1>5.3)
  {
   fprintf(ch7,"%f\n%f\n", d[0],d[1]);
   start++;
   sample=1;
   }
   if(start==1)ps[2]=RR_start[2]+i;
  if(start!=0 && sample==0)stop=1, pe[2]=RR_start[2]+i;
 }
  fclose(ch7);
   printf("temp_start_p3=%d\ttemp_end_p3=%d\n", ps[2], pe[2]);

 //cut pass4
d[2]=4.,d[3]=4.,d[4]=4.;
stop=0;
   if ((ch7 = fopen("ch7_p4.dat", "w+")) == NULL)
    { fprintf(stderr, "Cannot open file <ch7_p4.dat>.\n");
     exit(0);
    }
   for(start=0, i=0;i<200;i=i+2)
  {
  fscanf(fd7,"%f\n%f\n", d,d+1);
  }
  for(start=0,i=0;stop!=1;i=i+2)
  {
  sample=0;
   d[4]=d[2],d[2]=d[0],d[3]=d[1];
  fscanf(fd7,"%f\n%f\n", d,d+1);
  smt(5, d, dd);
   s1=( dd[0]+dd[1]+dd[2]+dd[3]+dd[4])/5.;
  if(dd[0]>4.2 && s1>4.2)
  {
   fprintf(ch7,"%f\n%f\n", d[0],d[1]);
   start++;
   sample=1;
   }
   if(start==1)ps[3]=pe[2]+i+200;
  if(start!=0 && sample==0)
    {  pe[3]=pe[2]+i+200;
    if((pe[3]-ps[3])<200){
    start=0;
    rewind (ch7);
    }
     else stop=1;
 }
}
   fclose(ch7);  printf("temp_start_p4=%d\ttemp_end_p4=%d\n", ps[3], pe[3]);

 //cut pass5
d[2]=4.0,d[3]=4.,d[4]=4.;
stop=0;
   if ((ch7 = fopen("ch7_p5.dat", "w+")) == NULL)
    { fprintf(stderr, "Cannot open file <ch7_p5.dat>.\n");
     exit(0);
    }
```

```
   for(start=0, i=pe[3];i<RR_start[4];i++)
 {
 fscanf(fd7,"%f\n", d);
 }
 for(start=0,i=0;stop!=1;i=i+2)
 {
 sample=0;
  d[4]=d[2],d[2]=d[0],d[3]=d[1];
 fscanf(fd7,"%f\n%f\n", d,d+1);
 smt(5, d, dd);
  s1=( dd[0]+dd[1]+dd[2]+dd[3]+dd[4])/5.;
 if(dd[0]>5.5 && s1>5.3)
 {
  fprintf(ch7,"%f\n%f\n", d[0],d[1]);
  start++;
  sample=1;
 }
   if(start==1)ps[4]=RR_start[4]+i;
 if(start!=0 && sample==0)stop=1, pe[4]=RR_start[4]+i;
}
 fclose(ch7);
  printf("temp_start_p5=%d\ttemp_end_p5=%d\n", ps[4], pe[4]);

 //cut pass6
d[2]=4.,d[3]=4.,d[4]=4.;
stop=0;
  if ((ch7 = fopen("ch7_p6.dat", "w+")) == NULL)
   { fprintf(stderr, "Cannot open file <ch7_p6.dat>.\n");
    exit(0);
  }
  for(start=0, i=0;i<200;i=i+2)
{ fscanf(fd7,"%f\n%f\n", d,d+1); }
 for(start=0,i=0;stop!=1;i=i+2)
{
 sample=0;
  d[4]=d[2],d[2]=d[0],d[3]=d[1];
 fscanf(fd7,"%f\n%f\n", d,d+1);
 smt(5, d, dd);
  s1=( dd[0]+dd[1]+dd[2]+dd[3]+dd[4])/5.;
 if(dd[0]>4.2 && s1>4.1)
 {
  fprintf(ch7,"%f\n%f\n", d[0],d[1]);
  start++;
  sample=1;
 }
   if(start==1)ps[5]=pe[4]+i+200;
 if(start!=0 && sample==0)
 {  pe[5]=pe[4]+i+200;
  if((pe[5]-ps[5])<250){
  start=0;
  rewind (ch7);
  }
   else stop=1;
}
}
 fclose(ch7);  printf("temp_start_p6=%d\ttemp_end_p6=%d\n", ps[5], pe[5]);

 //cut pass7
d[2]=4.0,d[3]=4.,d[4]=4.;
stop=0;
  if ((ch7 = fopen("ch7_p7.dat", "w+")) == NULL)
   { fprintf(stderr, "Cannot open file <ch7_p7.dat>.\n");
    exit(0);
  }
  for(start=0, i=pe[5];i<RR_start[6];i++)
{ fscanf(fd7,"%f\n", d); }
 for(start=0,i=0;stop!=1;i=i+2)
```

```
  {
   sample=0;
   d[4]=d[2],d[2]=d[0],d[3]=d[1];
  fscanf(fd7,"%f\n%f\n", d,d+1);
  smt(5, d, dd);
   s1=( dd[0]+dd[1]+dd[2]+dd[3]+dd[4])/5.;
  if(dd[0]>5.5 && s1>5.3)
  {
   fprintf(ch7,"%f\n%f\n", d[0],d[1]);
   start++;
   sample=1;
  }
    if(start==1)ps[6]=RR_start[6]+i;
  if(start!=0 && sample==0)
  {   pe[6]=RR_start[6]+i;
    if((pe[6]-ps[6])<200)
   {
    start=0;
    rewind (ch7);
   }
    else stop=1;
  }
 }
  fclose(ch7);
  fclose(fd7);
  printf("temp_start_p7=%d\ttemp_end_p7=%d\n", ps[6], pe[6]);

  return 0;
}// end of cut_chan_7()

//channel 8 tension is calculated
int cut_chan_8(int *ps,  int *p_e)
{
int i;
int start,stop=0,sample;
float d1;

   FILE *fd8;
   FILE  *ch8;
    if ((fd8 = fopen("ch8.dat", "r")) == NULL)
    { fprintf(stderr, "Cannot open file <ch8.dat>.\n");
     exit(0);
    }
   if ((ch8 = fopen("ch8_p1.dat", "w+")) == NULL)
    { fprintf(stderr, "Cannot open file <ch8_p1.dat>.\n");
     exit(0);
    }
//cut  pass1
 for(i=0;stop!=1;i++)
 {
 fscanf(fd8,"%f\n", &d1);
 if(i>ps[0])fprintf(ch8,"%f\n", d1);
  if(i==p_e[0])stop=1;
 }
 fclose(ch8);
  if ((ch8 = fopen("ch8_p3.dat", "w+")) == NULL)
   { fprintf(stderr, "Cannot open file <ch8_p3.dat>.\n");
    exit(0);
   }

//cut  pass3
 for(stop=0;stop!=1;i++)
 {
 fscanf(fd8,"%f\n", &d1);
 if(i>ps[2])fprintf(ch8,"%f\n", d1);
  if(i==p_e[2])stop=1;
 }
```

```
  fclose(ch8);
   if ((ch8 = fopen("ch8_p5.dat", "w+")) == NULL)
   { fprintf(stderr, "Cannot open file <ch8_p5.dat>.\n");
    exit(0);
   }

 //cut pass5
 for(stop=0;stop!=1;i++)
{
 fscanf(fd8,"%f\n", &d1);
 if(i> ps[4])fprintf(ch8,"%f\n", d1);
  if(i==p_e[4])stop=1;
}
 fclose(ch8);
  if ((ch8 = fopen("ch8_p7.dat", "w+")) == NULL)
   { fprintf(stderr, "Cannot open file <ch8_p7.dat>.\n");
    exit(0);
   }

//cut pass7
 for(stop=0;stop!=1;i++)
{
 fscanf(fd8,"%f\n", &d1);
 if(i> ps[6])fprintf(ch8,"%f\n", d1);
  if(i==p_e[6])stop=1;
}
 fclose(fd8);
 fclose(ch8);

return 0;
}

 /*cut the reversing pass of channel 4, pass 2, 4, 6. It considers time lag from RR to RE in reversing passes.*/.
int  cut_reverse(int *ps, int *pe)
{
 int ts,te;
 int i,j,t;
 float vr=0,va=0;
 float  chan4;
 FILE *fd4,*fd6;
 FILE *ch4;

   if ((fd4 = fopen("ch4.dat", "r")) == NULL)
   { fprintf(stderr, "Cannot open file <ch4.dat>.\n");
    exit(0);
   }
   if ((fd6 = fopen("ch6.dat", "r")) == NULL)
   { fprintf(stderr, "Cannot open file <ch6.dat>.\n");
    exit(0);
   }

//cut pass 2
   if ((ch4 = fopen("ch4_p2.dat", "w+")) == NULL)
   { fprintf(stderr, "Cannot open file <ch4_p2.dat>.\n");
    exit(0);
   }
for(i=0;i<ps[1];i++)
{
 fscanf(fd6, "%f\n",&vr);
 }
 for(t=0; t<10;t++,i++)
 {
  fscanf(fd6,"%f\n",&vr);
  va=vr+va;
  }
  va=va/10.;
 ts=350./(1.5*(4.-va));
```

```
 for(j=0;j<(ts+ps[1]);j++)
 {
 fscanf(fd4, "%f\n",&chan4);
 }

 for(;j<pe[1];j++)
 {
 fscanf(fd4, "%f\n",& chan4);
 fprintf(ch4,"%f\n", chan4);
  }
 for(t=0,va=0.;t<10;t++,j++)
 {
 fscanf(fd4, "%f\n",& chan4);
 fprintf(ch4,"%f\n", chan4);
  vr=chan4;
  va=vr+va;
  }
  va=va/10.;
 te=350./(1.5*(4.-va));
 for(;j<(pe[1]+te);j++)
 {
 fscanf(fd4, "%f\n",& chan4);
 fprintf(ch4,"%f\n", chan4);
 }

 fclose(ch4);
 printf("Reverse pass delay RR->RE p2 ts=%d\tte=%d\n",ts,te);

//cut  pass4
   if ((ch4 = fopen("ch4_p4.dat", "w+")) == NULL)
    { fprintf(stderr, "Cannot open file <ch4_p4.dat>.\n");
     exit(0);
   }
for(;i<ps[3];i++)
{
 fscanf(fd6, "%f\n",&vr);
}
 for(t=0,va=0.; t<10;t++,i++)
 {
  fscanf(fd6,"%f\n",&vr);
  va=vr+va;
 }
  va=va/10.;
 ts=350./(1.5*(4.-va));
for(;j<(ts+ps[3]);j++)
{
 fscanf(fd4, "%f\n",&chan4);
}

 for(;j<pe[3];j++)
 {
 fscanf(fd4, "%f\n",& chan4);
 fprintf(ch4,"%f\n", chan4);
  }
 for(t=0,va=0.;t<10;t++,j++)
 {
 fscanf(fd4, "%f\n",& chan4);
 fprintf(ch4,"%f\n", chan4);
  vr=chan4;
  va=vr+va;
  }
  va=va/10.;
 te=350./(1.5*(4.-va));
 for(;j<(pe[3]+te);j++)
 {
 fscanf(fd4, "%f\n",& chan4);
 fprintf(ch4,"%f\n", chan4);
```

```
  }
  fclose(ch4); printf("Reverse pass delay RR->RE p4 ts=%d\tte=%d\n",ts,te);

//cut pass6
  if ((ch4 = fopen("ch4_p6.dat", "w+")) == NULL)
   { fprintf(stderr, "Cannot open file <ch4_p6.dat>.\n");
    exit(0);
   }
for(;i<ps[5];i++)
{
fscanf(fd6, "%f\n",&vr);
}
 for(t=0,va=0.; t<10;t++,i++)
 {
  fscanf(fd6,"%f\n",&vr);
  va=vr+va;
 }
 va=va/10.;
 ts=350./(1.5*(4.-va));
for(;j<(ts+ps[5]);j++) fscanf(fd4, "%f\n",&chan4);
for(;j<pe[5];j++)
{
 fscanf(fd4, "%f\n",& chan4);
 fprintf(ch4,"%f\n", chan4);
  }
 for(t=0,va=0.;t<10;t++,j++)
{
 fscanf(fd4, "%f\n",& chan4);
 fprintf(ch4,"%f\n", chan4);
 vr=chan4;
 va=vr+va;
  }
 va=va/10.;
 te=350./(1.5*(4.-va));
 for(;j<(pe[5]+te);j++)
{
 fscanf(fd4, "%f\n",& chan4);
 fprintf(ch4,"%f\n", chan4);
 }
 fclose(ch4); fclose(fd4); fclose(fd6);
 printf("Reverse pass delay RR->RE p6 ts=%d\tte=%d\n",ts,te);
 return 0;
 }

/* This subroutine used to find out channel 1 signals, using time lag.*/
int cut_chan_1(int *pe, int *p1, int *p2)
{
int i,t;
FILE *fd1, *fd6,*ch1;
float d1=0;
float va=0;
int ts[]={0,0,0,0,0,0,0,0}, te[]={0,0,0,0,0,0,0,0};

   if ((fd1 = fopen("ch1.dat", "r")) == NULL)
   { fprintf(stderr, "Cannot open file <ch1.dat>.\n");
    exit(0);
   }
   if ((fd6 = fopen("ch6.dat", "r")) == NULL)
   { fprintf(stderr, "Cannot open file <ch6.dat>.\n");
    exit(0);
   }
//calculate pass1
 for(i=0;i<(p1[0]);i++) fscanf(fd6,"%f\n", &d1);
 for(t=0,va=0.; t<10;t++,i++)
 {
  fscanf(fd6,"%f\n",&d1);
  va=d1+va;
```

```
  }
   va=va/10.;
    ts[0]=50./(1.5*(va-4.));
 for(;i<(pe[0]-20);i++)fscanf(fd6, "%f\n",& d1);
 for(t=0,va=0.;t<20;t++,i++)
 {
  fscanf(fd6,"%f\n",&d1);
   va=d1+va;
   }
  va=va/20.;
  te[0]=50./(1.5*(va-4.));

  printf("channel 1 p1_start=%d\tp1_end=%d\n", p1[0]+ts[0],p2[0]+te[0]);

//cc pass2
 for(;i<(p1[1]-50);i++) fscanf(fd6,"%f\n", &d1);
 for(t=0,va=0.;t<30;t++,i++)
 {
  fscanf(fd6,"%f\n",&d1);
   va=d1+va;
  }
   va=va/30.;
    ts[1]=50./(1.5*(4.-va));
 for(;i<(p2[1]-30);i++)fscanf(fd6, "%f\n",& d1);
 for(t=0,va=0.;t<30;t++,i++)
 {
  fscanf(fd6,"%f\n",&d1);
   va=d1+va;
   }
  va=va/30.;
  te[1]=50./(1.5*(4.-va));
   printf("ch1 pass2_start1=%d\tp2_end=%d\n", p1[1]-ts[1],p2[1]-te[1]);

//calculate pass3
 for(;i<(p1[2]);i++) fscanf(fd6,"%f\n", &d1);
 for(t=0,va=0.; t<10;t++,i++)
 {
  fscanf(fd6,"%f\n",&d1);
   va=d1+va;
  }
   va=va/10.;
    ts[2]=50./(1.5*(va-4.));
 for(;i<(pe[2]-20);i++)fscanf(fd6, "%f\n",& d1);
 for(t=0,va=0.;t<20;t++,i++)
{
  fscanf(fd6,"%f\n",&d1);
   va=d1+va;
   }
  va=va/20.;
  te[2]=50./(1.5*(va-4.));

  printf("ch1 pass3_start=%d\tp3_end=%d\n", p1[2]+ts[2],p2[2]+te[2]);

//calculate pass4
 for(;i<(p1[3]-50);i++) fscanf(fd6,"%f\n", &d1);
 for(t=0,va=0.;t<30;t++,i++)
 {
  fscanf(fd6,"%f\n",&d1);
   va=d1+va;
  }
   va=va/30.;
    ts[3]=50./(1.5*(4.-va));
 for(;i<(p2[3]-30);i++)fscanf(fd6, "%f\n",& d1);
 for(t=0,va=0.;t<30;t++,i++)
{
  fscanf(fd6,"%f\n",&d1);
   va=d1+va;
```

```
   }
   va=va/30.;
   te[3]=50./(1.5*(4.-va));
   printf("ch1 pass4_start=%d\tp4_end=%d\n", p1[3]-ts[3],p2[3]-te[3]);
 //calculate pass5
 for(;i<(p1[4]);i++) fscanf(fd6,"%f\n", &d1);
 for(t=0,va=0.; t<10;t++,i++)
 {
  fscanf(fd6,"%f\n",&d1);
   va=d1+va;
  }
   va=va/10.;
   ts[4]=50./(1.5*(va-4.));
 for(;i<(pe[4]-20);i++)fscanf(fd6, "%f\n",& d1);
 for(t=0,va=0.;t<20;t++,i++)
 {
  fscanf(fd6,"%f\n",&d1);
   va=d1+va;
  }
  va=va/20.;
  te[4]=50./(1.5*(va-4.));
  printf("ch6 pass5_start=%d\tp5_end=%d\n", p1[4]+ts[4],p2[4]+te[4]);

//omit pass6, since pass 6 is noisy for channel 1.

//calculate pass7
 for(;i<(p1[6]);i++) fscanf(fd6,"%f\n", &d1);
 for(t=0,va=0.; t<10;t++,i++)
 {
  fscanf(fd6,"%f\n",&d1);
   va=d1+va;
  }
   va=va/10.;
   ts[6]=50./(1.5*(va-4.));
 for(;i<(pe[6]-20);i++)fscanf(fd6, "%f\n",& d1);
 for(t=0,va=0.;t<20;t++,i++)
 {
  fscanf(fd6,"%f\n",&d1);
   va=d1+va;
  }
  va=va/20.;
  te[6]=50./(1.5*(va-4.));
  printf("ch1 pass7_start=%d\tp7_end=%d\n", p1[6]+ts[6],p2[6]+te[6]);
 fclose(fd6);

//cut channel 1 and write pass1
   if ((ch1 = fopen("ch1_p1.dat", "w+")) == NULL)
   { fprintf(stderr, "Cannot open file <ch1_p1.dat>.\n");
    exit(0);
   }
 for(i=0;i<(p1[0]+ts[0]);i++)fscanf(fd1,"%f\n",&d1);
 for(;i<(p2[0]+te[0]);i++)
 {
  fscanf(fd1,"%f\n",&d1);
  fprintf(ch1,"%f\n", d1);
 }
fclose(ch1);

//write pass2
   if ((ch1 = fopen("ch1_p2.dat", "w+")) == NULL)
   { fprintf(stderr, "Cannot open file <ch1_p2.dat>.\n");
    exit(0);
   }
 for(;i<(p1[1]-ts[1]);i++)fscanf(fd1,"%f\n",&d1);
 for(;i<(p2[1]-te[1]);i++)
 {
  fscanf(fd1,"%f\n",&d1);
```

```
   fprintf(ch1,"%f\n", d1);
   }
 fclose(ch1);

 //write pass3
    if ((ch1 = fopen("ch1_p3.dat", "w+")) == NULL)
    { fprintf(stderr, "Cannot open file <ch1_p3.dat>.\n");
     exit(0);
   }
 for(;i<(p1[2]+ts[2]);i++)fscanf(fd1,"%f\n",&d1);
 for(;i<(p2[2]+te[2]);i++)
 {
  fscanf(fd1,"%f\n",&d1);
  fprintf(ch1,"%f\n", d1);
  }
 fclose(ch1);

 //write pass4
    if ((ch1 = fopen("ch1_p4.dat", "w+")) == NULL)
    { fprintf(stderr, "Cannot open file <ch1_p4.dat>.\n");
     exit(0);
   }
 for(;i<(p1[3]-ts[3]);i++)fscanf(fd1,"%f\n",&d1);
 for(;i<(p2[3]-te[3]);i++)
 {
  fscanf(fd1,"%f\n",&d1);
  fprintf(ch1,"%f\n", d1);
  }
 fclose(ch1);

 //write pass5
    if ((ch1 = fopen("ch1_p5.dat", "w+")) == NULL)
    { fprintf(stderr, "Cannot open file <ch1_p5.dat>.\n");
     exit(0);
   }
 for(;i<(p1[4]+ts[4]);i++)fscanf(fd1,"%f\n",&d1);
 for(;i<(p2[4]+te[4]);i++)
 {
  fscanf(fd1,"%f\n",&d1);
  fprintf(ch1,"%f\n", d1);
  }
 fclose(ch1);

 //write pass7
    if ((ch1 = fopen("ch1_p7.dat", "w+")) == NULL)
    { fprintf(stderr, "Cannot open file <ch1_p7.dat>.\n");
     exit(0);
   }
 for(;i<(p1[6]+ts[6]);i++)fscanf(fd1,"%f\n",&d1);
 for(;i<(p2[6]+te[6]);i++)
 {
  fscanf(fd1,"%f\n",&d1);
  fprintf(ch1,"%f\n", d1);
  }
 fclose(ch1);
 fclose(fd1);
 printf("finish ch1\n");
 return 0;

 }//end of cut_chan_1().

 /* This subroutine, also using time lag, figures out channel 0. It consider the time for width gauge 1 to RE.*/
 int cut_chan_0(int *ps, int *pe, int *p1, int *p2)
 {
 int i,j,t;
 FILE *fd0, *fd4,* fd6,*ch0;
 float d1=0,t1=0,t2=0;
```

```
float va=0;
int ts[]={0,0,0,0,0,0,0,0}, te[]={0,0,0,0,0,0,0,0};

   if ((fd6 = fopen("ch6.dat", "r")) == NULL)
    { fprintf(stderr, "Cannot open file <ch6.dat>.\n");
      exit(0);
    }
  if ((fd4 = fopen("ch4.dat", "r")) == NULL)
    { fprintf(stderr, "Cannot open file <ch4.dat>.\n");
      exit(0);
    }

//omit pass1, it is noise for channel 0 in pass 1.

//calculate pass2
 for(i=0;i<(ps[1]);i++) fscanf(fd6,"%f\n", &d1);
 t1=350./(1.5*(4.-d1));
 for(t=0,va=0.; t<t1;t++,i++)
 {
  fscanf(fd6,"%f\n",&d1);
   va=d1+va;
  }
   va=va/t;
    t1=350./(1.5*(4.-va));
 for(j=0;j<(ps[1]+t1);j++) fscanf(fd4,"%f\n", &d1);
 t2=300./(1.5*(4.-d1));
 for(t=0,va=0.; t<t2;t++,j++)
 {
  fscanf(fd4,"%f\n",&d1);
   va=d1+va;
  }
   va=va/t;
    t2=300./(1.5*(4.-va));
 ts[1]=t1+t2;
 for(;j<(pe[1]);j++)fscanf(fd4, "%f\n",& d1);
 t1=350./(1.5*(4.-d1));
 for(t=0,va=0.;t<t1;t++,j++)
 {
  fscanf(fd4,"%f\n",&d1);
   va=d1+va;
   }
  va=va/t;
  te[1]=650./(1.5*(4.-va));
  printf("ch0 pass2_start=%d\tp2_end=%d\n", ps[1]+ts[1],pe[1]+te[1]);

//calculate pass3
 for(;j<(p1[2]);j++) fscanf(fd4,"%f\n", &d1);
 for(t=0,va=0.; t<30;t++,i++)
 {
  fscanf(fd4,"%f\n",&d1);
   va=d1+va;
  }
  va=va/t;
   t1=300./(1.5*(va-4.));
 ts[2]=t1;
 for(;j<(p2[2]);j++)fscanf(fd4, "%f\n",& d1);
 t1=350./(1.5*(d1-4.));
 rewind(fd4);
 for(j=0;j<(p2[2]-t1);j++)fscanf(fd4,"%f\n",&d1);
 for(t=0,va=0.;t<t1;t++,j++)
 {
  fscanf(fd4,"%f\n",&d1);
   va=d1+va;
  }
  va=va/t;
  te[2]=350./(1.5*(va-4.));
  printf("ch0 pass3_start=%d\tp3_end=%d\n", p1[2]-ts[2],p2[2]-te[2]);
```

```
//calculate pass4
 for(;i<(ps[3]);i++) fscanf(fd6,"%f\n", &d1);
 t1=350./(1.5*(4.-d1));
 for(t=0,va=0.; t<t1;t++,i++)
 {
  fscanf(fd6,"%f\n",&d1);
   va=d1+va;
  }
   va=va/t;
    t1=350./(1.5*(4.-va));
 for(;j<(ps[3]+t1);j++) fscanf(fd4,"%f\n", &d1);
 t2=300./(1.5*(4.-d1));
 for(t=0,va=0.; t<t2;t++,j++)
 {
  fscanf(fd4,"%f\n",&d1);
   va=d1+va;
  }
   va=va/t;
    t2=300./(1.5*(4-va));
 ts[3]=t1+t2;
 for(;j<(pe[3]);j++)fscanf(fd4, "%f\n",& d1);
 t1=350./(1.5*(4.-d1));
 for(t=0,va=0.;t<t1;t++,j++)
 {
  fscanf(fd4,"%f\n",&d1);
   va=d1+va;
   }
  va=va/t;
   te[3]=650./(1.5*(4.-va));
   printf("ch0 pass4_start=%d\tp4_end=%d\n", ps[3]+ts[3],pe[3]+te[3]);

// calculate pass5
 for(;j<(p1[4]);j++) fscanf(fd4,"%f\n", &d1);
 for(t=0,va=0.; t<30;t++,i++)
 {
  fscanf(fd4,"%f\n",&d1);
   va=d1+va;
  }
  va=va/t;
    t1=300./(1.5*(va-4.));
 ts[4]=t1;
 for(;j<(p2[4]);j++)fscanf(fd4, "%f\n",& d1);
 t1=350./(1.5*(d1-4.));
 rewind(fd4);

 for(j=0;j<(p2[4]-t1);j++)fscanf(fd4,"%f\n",&d1);
 for(t=0,va=0;t<t1;t++,j++)
 {
  fscanf(fd4,"%f\n",&d1);
   va=d1+va;
   }
  va=va/t;
  te[4]=350./(1.5*(va-4.));
   printf("ch0 pass5_start=%d\tp5_end=%d\n", p1[4]-ts[4],p2[4]-te[4]);

// calculate pass6
 for(;i<(ps[5]);i++) fscanf(fd6,"%f\n", &d1);
 t1=350./(1.5*(4.-d1));
 for(t=0,va=0; t<t1;t++,i++)
 {
  fscanf(fd6,"%f\n",&d1);
   va=d1+va;
  }
   va=va/t;
    t1=350./(1.5*(4.-va));
 for(;j<(ps[5]+t1);j++) fscanf(fd4,"%f\n", &d1);
```

```
  t2=300./(1.5*(4.-d1));
  for(t=0,va=0.; t<t2;t++,j++)
  {
   fscanf(fd4,"%f\n",&d1);
    va=d1+va;
   }
    va=va/t;
     t2=300./(1.5*(4.-va));
  ts[5]=t1+t2;

  for(;j<(pe[5]);j++)fscanf(fd4, "%f\n",& d1);
  t1=350/(1.5*(4.-d1));
  for(t=0,va=0.;t<t1;t++,j++)
  {
   fscanf(fd4,"%f\n",&d1);
    va=d1+va;
    }
   va=va/t;
    te[5]=650./(1.5*(4.-va));
    printf("ch0 pass6_start=%d\tp6_end=%d\n", ps[5]+ts[5],pe[5]+te[5]);

 // calculate pass7
  for(;j<(p1[6]);j++) fscanf(fd4,"%f\n", &d1);
  for(t=0,va=0.; t<30;t++,i++)
  {
   fscanf(fd4,"%f\n",&d1);
    va=d1+va;
   }
   va=va/t;
     t1=300./(1.5*(va-4.));
  ts[6]=t1;
  for(;j<(p2[6]);j++)fscanf(fd4, "%f\n",& d1);
  t1=350./(1.5*(d1-4.));
  rewind(fd4);
  for(j=0;j<(p2[6]-t1);j++)fscanf(fd4,"%f\n",&d1);
  for(t=0,va=0.;t<t1;t++,j++)
  {
   fscanf(fd4,"%f\n",&d1);
    va=d1+va;
    }
   va=va/t;
    te[6]=350./(1.5*(va-4.));
    printf("ch0 pass7_start=%d\tp7_end=%d\n", p1[6]-ts[6],p2[6]-te[6]);

 fclose(fd4);
 fclose(fd6);

 //start write channel 0
    if ((fd0 = fopen("ch0.dat", "r")) == NULL)
    { fprintf(stderr, "Cannot open file <ch0.dat>.\n");
      exit(0);
    }
 //write  pass2
    if ((ch0 = fopen("ch0_p2.dat", "w+")) == NULL)
    { fprintf(stderr, "Cannot open file <ch0_p2.dat>.\n");
      exit(0);
    }

  for(i=0;i<(ps[1]+ts[1]);i++)fscanf(fd0,"%f\n",&d1);
  for(;i<(pe[1]+te[1]);i++)
  {
   fscanf(fd0,"%f\n",&d1);
   fprintf(ch0,"%f\n", d1);
  }
 fclose(ch0);
 //write  pass3
    if ((ch0 = fopen("ch0_p3.dat", "w+")) == NULL)
```

```
    { fprintf(stderr, "Cannot open file <ch0_p3.dat>.\n");
     exit(0);
    }

 for(;i<(p1[2]-ts[2]);i++)fscanf(fd0,"%f\n",&d1);
 for(;i<(p2[2]-te[2]);i++)
 {
  fscanf(fd0,"%f\n",&d1);
  fprintf(ch0,"%f\n", d1);
 }
fclose(ch0);

//write pass4
   if ((ch0 = fopen("ch0_p4.dat", "w+")) == NULL)
    { fprintf(stderr, "Cannot open file <ch0_p4.dat>.\n");
     exit(0);
    }
 for(;i<(ps[3]+ts[3]);i++)fscanf(fd0,"%f\n",&d1);
 for(;i<(pe[3]+te[3]);i++)
 {
  fscanf(fd0,"%f\n",&d1);
  fprintf(ch0,"%f\n", d1);
 }
fclose(ch0);
//write pass5
   if ((ch0 = fopen("ch0_p5.dat", "w+")) == NULL)
    { fprintf(stderr, "Cannot open file <ch0_p5.dat>.\n");
     exit(0);
    }
 for(;i<(p1[4]-ts[4]);i++)fscanf(fd0,"%f\n",&d1);
 for(;i<(p2[4]-te[4]);i++)
 {
  fscanf(fd0,"%f\n",&d1);
  fprintf(ch0,"%f\n", d1);
 }
fclose(ch0);

//write pass6
   if ((ch0 = fopen("ch0_p6.dat", "w+")) == NULL)
    { fprintf(stderr, "Cannot open file <ch0_p6.dat>.\n");
     exit(0);
    }
 for(;i<(ps[5]+ts[5]);i++)fscanf(fd0,"%f\n",&d1);
 for(;i<(pe[5]+te[5]);i++)
 {
  fscanf(fd0,"%f\n",&d1);
  fprintf(ch0,"%f\n", d1);
 }
fclose(ch0);

//write pass7
   if ((ch0 = fopen("ch0_p7.dat", "w+")) == NULL)
    { fprintf(stderr, "Cannot open file <ch0_p7.dat>.\n");
     exit(0);
    }
 for(;i<(p1[6]-ts[6]);i++)fscanf(fd0,"%f\n",&d1);
 for(;i<(p2[6]-te[6]);i++)
 {
  fscanf(fd0,"%f\n",&d1);
  fprintf(ch0,"%f\n", d1);
 }
fclose(ch0);fclose(fd0);
return 0;
} // end of cut_chan_0;
//end of file: sub2.cpp.
```

/*file: sub3, start*/

```
#include <stdio.h>
#include <graphics.h>
#include <math.h>
#include <stdlib.h>
#include <string.h>

 /*make files for the NN training bar-to-bar model and inbar model of different pass*/
 /*pass 2: BAR-BAR , Head 1.5m, tail 1.5m,  full file for the NN training model*/
int bb_p2(char *fname,int t2, float length)
{
  FILE *fp1, *fp2;
  FILE *fd1, *fd6, *fd0;
  float cha1, cha6, cha0;
  float sum;
  int i,j;
  char file[30];

  strcpy(file,fname);
   strncat(file,".bh2", 5);
if ((fp1 = fopen(file, "w+")) == NULL)
  {   fprintf(stderr, "Cannot open head file.\n");
   exit  (0);}
   strcpy(file,fname);
   strncat(file,".bt2", 5);

if ((fp2 = fopen(file, "w+")) == NULL)
  {
   fprintf(stderr, "Cannot open tail file.\n");
   exit  (0);
  }
//open divided channel data file
 if ((fd0 = fopen("ch0_p2.dat", "r")) == NULL)
  { fprintf(stderr, "Cannot open file <ch0_p2.dat>.\n");
   exit(0);
  }
 if ((fd1 = fopen("ch1_p2.dat", "r")) == NULL)
   {  fprintf(stderr, "Cannot open file <ch1_p2.dat>.\n");
    exit(0);
   }
   if ((fd6 = fopen("ch6_p2.dat", "r")) == NULL)
    { fprintf(stderr, "Cannot open file <ch6_p2.dat>.\n");
     exit(0);
    }
 // estimation for head and tail
 i=t2;
for(sum=0,j=0;j<10;j++)
 {
  fscanf(fd6, "%f\n", &cha6);
  sum=cha6+sum;
  }
  sum=sum/10.;
int t1=length*100./(1.5*(4.0-sum));  //with 1.5m for head
 t2=t2-t1;
 printf("bb t1=%d, t2=%d,", t1, t2);
 // make channel data files into training files, cut body
  for(j=0;j<t2;j++)
  {
   fscanf(fd0, "%f\n", &cha0);
   fscanf(fd1, "%f\n", &cha1);
   if (j<t1)
   {
   fprintf(fp1,"%.3f\t%.3f\n",cha0,cha1);
   }
  }
   for(;j<i;j++)
   {
    fscanf(fd0,"%f\n",&cha0);
```

```
 fscanf(fd1,"%f\n",&cha1);
 fprintf(fp2, "%.3f\t%.3f\n",cha0,cha1);
 }
 fclose(fd0); fclose(fd1); fclose(fp1);
 fclose(fp2); fclose(fd6);
 return 0;
}//end of bb_p2().

/*pass 4: BAR-BAR , Head 1.5m, tail 1.5m,
   full file for the NN training model*/
int bb_p4(char *fname,int t2, float length)
{
 FILE *fp1, *fp2;
 FILE *fd1, *fd6, *fd0;
 float cha1, cha6, cha0;

 float sum;
 int i,j;
 char file[30];
 strcpy(file,fname);
  strncat(file,".bh4", 5);
if ((fp1 = fopen(file, "w+")) == NULL)
 {
  fprintf(stderr, "Cannot open head file.\n");
  exit (0);
  }
  strcpy(file,fname);
  strncat(file,".bt4", 5);
if ((fp2 = fopen(file, "w+")) == NULL)
 {
  fprintf(stderr, "Cannot open tail file.\n");
  exit (0);
  }
//open divided channel data file
 if ((fd0 = fopen("ch0_p4.dat", "r")) == NULL)
  { fprintf(stderr, "Cannot open file <ch0_p4.dat>.\n");
  exit(0);
  }
 if ((fd1 = fopen("ch1_p4.dat", "r")) == NULL)
   {  fprintf(stderr, "Cannot open file <ch1_p4.dat>.\n");
   exit(0);
  }
   if ((fd6 = fopen("ch6_p4.dat", "r")) == NULL)
    { fprintf(stderr, "Cannot open file <ch6_p4.dat>.\n");
     exit(0);
     }
 // estimation for head and tail
 i=t2;
for(sum=0,j=0;j<10;j++)
 {
 fscanf(fd6, "%f", &cha6);
 sum=cha6+sum;
 }
 sum=sum/10.;
int  t1=length*100./(1.5*(4.0-sum));  //with 1.5m for head
 t2=i-t1;

 // make channel data files into training files, cut body
  for(j=0;j<t1;j++)
  {
  fscanf(fd0, "%f", &cha0);
  fscanf(fd1, "%f", &cha1);
  fprintf(fp1,"%.3f\t%.3f\n",cha0,cha1);
  }
  for(;j<t2;j++)
  {
  fscanf(fd0, "%f", &cha0);
```

```
  fscanf(fd1, "%f", &cha1);
  }
  for(;j<i;j++)
  {
  fscanf(fd0,"%.3f",&cha0);
  fscanf(fd1,"%.3f",&cha1);
  fprintf(fp2, "%.3f\t%.3f\n",cha0,cha1);
  }
  fclose(fd0); fclose(fd1); fclose(fp1); fclose(fp2);
  fclose(fd6);
 return 0;
}//end of bb4().

/*In-bar body: Pass 3, substract head and tail part*/
int in_p3(char *fname,int t1,int t2, float length)
{
 FILE *fp1;
 FILE *fd1, *fd2, *fd3, *fd4, *fd5, *fd6, *fd7, *fd8, *fd9, *fd10,\
    *fd11, *fd12, *fd0;
 float cha1, cha2, cha3, cha4, cha5 ,cha6, cha7, cha8, cha9, cha10, cha11,
          cha12, cha0;
 float sum;
  int i,j,read_pointer=0;
 char ch;
  char file[30];
 strcpy(file,fname);
 strncat(file,".in3", 5);

if ((fp1 = fopen(file, "w+")) == NULL)
 {
  fprintf(stderr, "Cannot open training file.\n");
  exit (0);
  }
//open divided channel data file
 if ((fd0 = fopen("ch0_p3.dat", "r")) == NULL)
  { fprintf(stderr, "Cannot open file <ch0_p3.dat>.\n");
   exit(0);
  }
 if ((fd1 = fopen("ch1_p3.dat", "r")) == NULL)
   {  fprintf(stderr, "Cannot open file <ch1_p3.dat>.\n");
    exit(0);
  }
  if ((fd2 = fopen("ch2_p3.dat", "r")) == NULL)
    { fprintf(stderr, "Cannot open file <ch2_p3.dat>.\n");
    exit(0);
   }
  if ((fd3 = fopen("ch3_p3.dat", "r")) == NULL)
    { fprintf(stderr, "Cannot open file <ch3_p3.dat>.\n");
    exit(0);
    }
  if ((fd4 = fopen("ch4_p3.dat", "r")) == NULL)
     { fprintf(stderr, "Cannot open file <ch4_p3.dat>.\n");
      exit(0);
     }
  if ((fd5 = fopen("ch5_p3.dat", "r")) == NULL)
 { fprintf(stderr, "Cannot open file <ch5_p3.dat>.\n");
     exit(0);
    }
  if ((fd6 = fopen("ch6_p3.dat", "r")) == NULL)
    { fprintf(stderr, "Cannot open file <ch6_p3.dat>.\n");
     exit(0);
   }
  if ((fd7 = fopen("ch7_p3.dat", "r")) == NULL)
    { fprintf(stderr, "Cannot open file <ch7_p3.dat>.\n");
   exit(0);
    }
  if ((fd8 = fopen("ch8_p3.dat", "r")) == NULL)
```

```
    { fprintf(stderr, "Cannot open file <ch8_p3.dat>.\n");
    exit(0);
    }
   if ((fd9 = fopen("ch9_p3.dat", "r")) == NULL)
    { fprintf(stderr, "Cannot open file <ch9_p3.dat>.\n");
    exit(0);
    }
   if ((fd10 = fopen("ch10_p3.dat", "r")) == NULL)
    { fprintf(stderr, "Cannot open file <ch10_p3.dat>.\n");
          exit(0);
    }
   if ((fd11 = fopen("ch11_p3.dat", "r")) == NULL)
    { fprintf(stderr, "Cannot open file <ch11_p3.dat>.\n");
          exit(0);
    }
   if ((fd12 = fopen("ch12_p3.dat", "r")) == NULL)
    { fprintf(stderr, "Cannot open file <ch12_p3.dat>.\n");
          exit(0);
    }
 // estimation for head and tail
  i=(t1>t2)?t2:t1;
for(sum=0,j=0;j<10;j++)
 {
  fscanf(fd4, "%f", &cha4);
  sum=cha4+sum;
 }
  sum=sum/10.;
 t1=length*100./(1.5*(sum-4.0));  //with head
 t2=i-t1;
 rewind(fd4);
 // make channel data files into one training file, cut head and tail
  for(i=1;i<t2;i++)
 {
  fscanf(fd0, "%f", &cha0);
  fscanf(fd1, "%f", &cha1);
  fscanf(fd2, "%f", &cha2);
  fscanf(fd3, "%f", &cha3);
  fscanf(fd4, "%f", &cha4);
  fscanf(fd5, "%f", &cha5);
  fscanf(fd6, "%f", &cha6);
  fscanf(fd7, "%f", &cha7);
  fscanf(fd9, "%f", &cha9);
  fscanf(fd10, "%f", &cha10);
  fscanf(fd11, "%f", &cha11);
  fscanf(fd12, "%f", &cha12);
  if (i>=t1)
  {
    if(read_pointer==1) cha8=4;
   else fscanf(fd8, "%f", &cha8);
  fprintf(fp1,"%.3f\t%.3f\t%.3f\t%.3f\t%.3f\t%.3f\t%.3f\t%.3f\t%.3f\t%.3f\t%.3f\t%.3f\t%.3f\t%.3f\n",\
   cha0,cha2,cha3,cha4,cha5,cha6,cha7,cha8,cha9,cha10,cha11,cha12,cha1);
   ch=fgetc(fd8);
    if(ch==EOF)read_pointer=1;
  }
 }
   printf("t1=%d\nt2=%d\n",t1,t2);
 fclose(fd0); fclose(fd1); fclose(fd2); fclose(fd3);
 fclose(fd4); fclose(fd5); fclose(fd6); fclose(fd7);
 fclose(fd8); fclose(fd9); fclose(fd10); fclose(fd11);
 fclose(fd12); fclose (fp1);
 return 0;
}//end of subrountine in_p3().

/*In-bar body: Pass 5, substract head and tail part*/
int in_p5(char *fname,int t1,int t2, float length)
{
 FILE *fp1;
```

```
 FILE *fd1, *fd2, *fd3, *fd4, *fd5, *fd6, *fd7, *fd8, *fd9, *fd10,\
    *fd11, *fd12, *fd0;
 float cha1, cha2, cha3, cha4, cha5 ,cha6, cha7, cha8, cha9, cha10, cha11,
         cha12, cha0;
 float sum;
  int i,j,read_pointer=0;
 char ch, file[30];
 strcpy(file,fname);
 strncat(file,".in5", 5);

if ((fp1 = fopen(file, "w+")) == NULL)
 {
  fprintf(stderr, "Cannot open training file.\n");
  exit  (0);
 }
//open divided channel data file
 if ((fd0 = fopen("ch0_p5.dat", "r")) == NULL)
  { fprintf(stderr, "Cannot open file <ch0_p5.dat>.\n");
  exit(0);
  }
 if ((fd1 = fopen("ch1_p5.dat", "r")) == NULL)
   {  fprintf(stderr, "Cannot open file <ch1_p5.dat>.\n");
   exit(0);
  }
  if ((fd2 = fopen("ch2_p5.dat", "r")) == NULL)
    { fprintf(stderr, "Cannot open file <ch2_p5.dat>.\n");
    exit(0);
   }
  if ((fd3 = fopen("ch3_p5.dat", "r")) == NULL)
    { fprintf(stderr, "Cannot open file <ch3_p5.dat>.\n");
    exit(0);
    }
  if ((fd4 = fopen("ch4_p5.dat", "r")) == NULL)
     { fprintf(stderr, "Cannot open file <ch4_p5.dat>.\n");
      exit(0);
     }
  if ((fd5 = fopen("ch5_p5.dat", "r")) == NULL)
   { fprintf(stderr, "Cannot open file <ch5_p5.dat>.\n");
    exit(0);
   }
  if ((fd6 = fopen("ch6_p5.dat", "r")) == NULL)
   { fprintf(stderr, "Cannot open file <ch6_p5.dat>.\n");
    exit(0);
  }
  if ((fd7 = fopen("ch7_p5.dat", "r")) == NULL)
    { fprintf(stderr, "Cannot open file <ch7_p5.dat>.\n");
   exit(0);
   }
  if ((fd8 = fopen("ch8_p5.dat", "r")) == NULL)
    { fprintf(stderr, "Cannot open file <ch8_p5.dat>.\n");
    exit(0);
   }
  if ((fd9 = fopen("ch9_p5.dat", "r")) == NULL)
    { fprintf(stderr, "Cannot open file <ch9_p5.dat>.\n");
    exit(0);
    }
   if ((fd10 = fopen("ch10_p5.dat", "r")) == NULL)
    {  fprintf(stderr, "Cannot open file <ch10_p5.dat>.\n");
           exit(0);
    }
   if ((fd11 = fopen("ch11_p5.dat", "r")) == NULL)
     { fprintf(stderr, "Cannot open file <ch11_p5.dat>.\n");
            exit(0);
    }
   if ((fd12 = fopen("ch12_p5.dat", "r")) == NULL)
     { fprintf(stderr, "Cannot open file <ch12_p5.dat>.\n");
           exit(0);
```

```
   }
 // estimation for head and tail
  i=(t1>t2)?t2:t1;
for(sum=0,j=0;j<10;j++)
 {
 fscanf(fd4, "%f", &cha4);
 sum=cha4+sum;
 }
 sum=sum/10.;
 t1=length*100./(1.5*(sum-4.0));  //with head
 t2=i-t1;
 rewind(fd4);

 // make channel data files into one training file, cut head and tail
  for(i=1;i<t2;i++)
 {
  fscanf(fd0, "%f", &cha0);
  fscanf(fd1, "%f", &cha1);
  fscanf(fd2, "%f", &cha2);
  fscanf(fd3, "%f", &cha3);
  fscanf(fd4, "%f", &cha4);
  fscanf(fd5, "%f", &cha5);
  fscanf(fd6, "%f", &cha6);
  fscanf(fd7, "%f", &cha7);
  fscanf(fd9, "%f", &cha9);
  fscanf(fd10, "%f", &cha10);
  fscanf(fd11, "%f", &cha11);
  fscanf(fd12, "%f", &cha12);

  if (i>=t1)
  {
    if(read_pointer==1) cha8=4;
   else fscanf(fd8, "%f", &cha8);
  fprintf(fp1,"%.3f\t%.3f\t%.3f\t%.3f\t%.3f\t%.3f\t%.3f\t%.3f\t%.3f\t%.3f\t%.3f\t%.3f\t%.3f\t%.3f\t%.3f\n",\
   cha0,cha2,cha3,cha4,cha5,cha6,cha7,cha8,cha9,cha10,cha11,cha12,cha1);
   ch=fgetc(fd8);
    if(ch==EOF)read_pointer=1;
  }
 }
   printf("t1=%d\nt2=%d\n",t1,t2);
 fclose(fd0);  fclose(fd1);  fclose(fd2);  fclose(fd3);
 fclose(fd4);  fclose(fd5);  fclose(fd6);  fclose(fd7);
 fclose(fd8);  fclose(fd9);  fclose(fd10);  fclose(fd11);
 fclose(fd12);  fclose (fp1);
 return 0;
}// end of in_p5().

/*In-bar body: Pass 7, substract head and tail part*/
int in_p7(char *fname,int t1,int t2, float length)
{
 FILE *fp1;
 FILE *fd1, *fd2, *fd3, *fd4, *fd5, *fd6, *fd7, *fd8, *fd9, *fd10,\
    *fd11, *fd12, *fd0;
 float cha1, cha2, cha3, cha4, cha5 ,cha6, cha7, cha8, cha9, cha10, cha11,
          cha12, cha0;
 float sum;
  int i,j,read_pointer=0;
 char ch,file[30];
 strcpy(file,fname);
 strncat(file,".in7", 5);

if ((fp1 = fopen(file, "w+")) == NULL)
 {
  fprintf(stderr, "Cannot open training file.\n");
  exit  (0);
  }
//open divided channel data file
```

```
if ((fd0 = fopen("ch0_p7.dat", "r")) == NULL)
 { fprintf(stderr, "Cannot open file <ch0_p7.dat>.\n");
  exit(0);
 }
if ((fd1 = fopen("ch1_p7.dat", "r")) == NULL)
  {  fprintf(stderr, "Cannot open file <ch1_p7.dat>.\n");
   exit(0);
 }
 if ((fd2 = fopen("ch2_p7.dat", "r")) == NULL)
   { fprintf(stderr, "Cannot open file <ch2_p7.dat>.\n");
   exit(0);
  }
 if ((fd3 = fopen("ch3_p7.dat", "r")) == NULL)
   { fprintf(stderr, "Cannot open file <ch3_p7.dat>.\n");
   exit(0);
   }
  if ((fd4 = fopen("ch4_p7.dat", "r")) == NULL)
    { fprintf(stderr, "Cannot open file <ch4_p7.dat>.\n");
     exit(0);
    }
  if ((fd5 = fopen("ch5_p7.dat", "r")) == NULL)
   { fprintf(stderr, "Cannot open file <ch5_p7.dat>.\n");
    exit(0);
   }
  if ((fd6 = fopen("ch6_p7.dat", "r")) == NULL)
   { fprintf(stderr, "Cannot open file <ch6_p7.dat>.\n");
    exit(0);
  }
  if ((fd7 = fopen("ch7_p7.dat", "r")) == NULL)
    { fprintf(stderr, "Cannot open file <ch7_p7.dat>.\n");
   exit(0);
   }
  if ((fd8 = fopen("ch8_p7.dat", "r")) == NULL)
    { fprintf(stderr, "Cannot open file <ch8_p7.dat>.\n");
    exit(0);
   }
  if ((fd9 = fopen("ch9_p7.dat", "r")) == NULL)
    { fprintf(stderr, "Cannot open file <ch9_p7.dat>.\n");
    exit(0);
    }
   if ((fd10 = fopen("ch10_p7.dat", "r")) == NULL)
    {  fprintf(stderr, "Cannot open file <ch10_p7.dat>.\n");
          exit(0);
    }
   if ((fd11 = fopen("ch11_p7.dat", "r")) == NULL)
     { fprintf(stderr, "Cannot open file <ch11_p7.dat>.\n");
          exit(0);
    }
   if ((fd12 = fopen("ch12_p7.dat", "r")) == NULL)
     { fprintf(stderr, "Cannot open file <ch12_p7.dat>.\n");
          exit(0);
    }
 // estimation for head and tail
  i=(t1>t2)?t2:t1;
for(sum=0,j=0;j<10;j++)
{
 fscanf(fd4, "%f", &cha4);
 sum=cha4+sum;
}
 sum=sum/10.;
 t1=length*100./(1.5*(sum-4.0));   //with head
 t2=i-t1;
 rewind(fd4);
 // make channel data files into one training file, cut head and tail
  for(i=1;i<t2;i++)
 {
  fscanf(fd0, "%f", &cha0);  fscanf(fd1, "%f", &cha1);
```

```
  fscanf(fd2, "%f", &cha2);  fscanf(fd3, "%f", &cha3);
  fscanf(fd4, "%f", &cha4);  fscanf(fd5, "%f", &cha5);
  fscanf(fd6, "%f", &cha6);  fscanf(fd7, "%f", &cha7);
  fscanf(fd9, "%f", &cha9);  fscanf(fd10, "%f", &cha10);
  fscanf(fd11, "%f", &cha11);  fscanf(fd12, "%f", &cha12);

  if (i>=t1)
  {
    if(read_pointer==1) cha8=4;
   else fscanf(fd8, "%f", &cha8);
  fprintf(fp1,"%.3f\t%.3f\t%.3f\t%.3f\t%.3f\t%.3f\t%.3f\t%.3f\t%.3f\t%.3f\t%.3f\t%.3f\t%.3f\t%.3f\n",\
   cha0,cha2,cha3,cha4,cha5,cha6,cha7,cha8,cha9,cha10,cha11,cha12,cha1);
  ch=fgetc(fd8);
   if(ch==EOF)read_pointer=1;
  }
 }
   printf("t1=%d\nt2=%d\n",t1,t2);
 fclose(fd0);  fclose(fd1);  fclose(fd2);  fclose(fd3);
 fclose(fd4);  fclose(fd5);  fclose(fd6);  fclose(fd7);
 fclose(fd8);  fclose(fd9);  fclose(fd10);  fclose(fd11);
 fclose(fd12);  fclose (fp1);
 return 0;
}

/* delete all the files generated in  the processing, which is not needed for neural networks*/
int clean()
{
 unlink("ch0.dat");  unlink("ch1.dat");  unlink("ch2.dat");  unlink("ch3.dat");
 unlink("ch4.dat");  unlink("ch5.dat");  unlink("ch6.dat");  unlink("ch7.dat");
 unlink("ch8.dat");  unlink("ch9.dat");  unlink("ch10.dat");  unlink("ch11.dat");  unlink("ch12.dat");

unlink("ch0_p2.dat");  unlink("ch0_p3.dat");  unlink("ch0_p4.dat");  unlink("ch0_p5.dat");
  unlink("ch0_p6.dat");  unlink("ch0_p7.dat");
  unlink("ch1_p1.dat");  unlink("ch1_p2.dat");  unlink("ch1_p3.dat");  unlink("ch1_p4.dat");
  unlink("ch1_p5.dat");  unlink("ch1_p7.dat");
  unlink("ch2_p1.dat");  unlink("ch2_p3.dat");  unlink("ch2_p5.dat");  unlink("ch2_p7.dat");
  unlink("ch3_p1.dat");  unlink("ch3_p3.dat");  unlink("ch3_p5.dat");  unlink("ch3_p7.dat");
  unlink("ch4_p1.dat");  unlink("ch4_p2.dat");  unlink("ch4_p3.dat");  unlink("ch4_p4.dat");
  unlink("ch4_p5.dat");  unlink("ch4_p6.dat");  unlink("ch4_p7.dat");
  unlink("ch5_p1.dat");  unlink("ch5_p3.dat");  unlink("ch5_p5.dat");  unlink("ch5_p7.dat");
  unlink("ch6_p1.dat");  unlink("ch6_p2.dat");  unlink("ch6_p3.dat");  unlink("ch6_p4.dat");
  unlink("ch6_p5.dat");  unlink("ch6_p6.dat");  unlink("ch6_p7.dat");
  unlink("ch7_p1.dat");  unlink("ch7_p2.dat");  unlink("ch7_p3.dat");  unlink("ch7_p4.dat");
  unlink("ch7_p5.dat");  unlink("ch7_p6.dat");  unlink("ch7_p7.dat");
  unlink("ch8_p1.dat");  unlink("ch8_p3.dat");  unlink("ch8_p5.dat");  unlink("ch8_p7.dat");
  unlink("ch9_p1.dat");  unlink("ch9_p2.dat");  unlink("ch9_p3.dat");  unlink("ch9_p4.dat");
  unlink("ch9_p5.dat");  unlink("ch9_p6.dat");  unlink("ch9_p7.dat");
  unlink("ch10_p1.dat");  unlink("ch10_p2.dat");  unlink("ch10_p3.dat");  unlink("ch10_p4.dat");
  unlink("ch10_p5.dat");  unlink("ch10_p6.dat");  unlink("ch10_p7.dat");
unlink("ch11_p1.dat");  unlink("ch11_p2.dat");  unlink("ch11_p3.dat");  unlink("ch11_p4.dat");
unlink("ch11_p5.dat");  unlink("ch11_p6.dat");  unlink("ch11_p7.dat");
  unlink("ch12_p1.dat");  unlink("ch12_p2.dat");  unlink("ch12_p3.dat");  unlink("ch12_p4.dat");
  unlink("ch12_p5.dat");  unlink("ch12_p6.dat");  unlink("ch12_p7.dat");
 return 0;

}// end of clean().

// end of file sub3.cpp.
```

D.B. Aitchison
Bookbinder
122 Windang Rd.
Primbee 2502
Ph (042) 742229